Heurísticas y Metaconocimiento en Resolución Automática de Problemas de Matemáticas

Autor: Pablo Castells Azpilicueta
Director: Roberto Moriyón Salomón

Departamento de Ingeniería Informática
Facultad de Ciencias
Universidad Autónoma de Madrid

Marzo 1994

Memoria presentada para optar al grado de doctor en Ingeniería Informática

A Ana

Agradecimientos

Esta tesis no habría sido posible sin la ayuda, apoyo, paciencia y dedicación constante de Roberto Moriyón, director de la misma y coordinador del proyecto PROGENES, en cuyo contexto ha tenido lugar la elaboración de este trabajo. Su apoyo y su ánimo han estado presentes en todo momento.

Quiero expresar también mi agradecimiento a Francisco Saiz, que ha sido un gran compañero, no sólo en el trabajo diario, sino también en las experiencias y vicisitudes de estos años. Gracias también a Julio Gonzalo por su contagioso entusiasmo y por conseguir motivar mi interés hacia el Procesamiento del Lenguaje Natural.

Debo agradecer asímismo a Dominique Pastre su presencia en el tribunal de esta tesis, así como su colaboración en el desarrollo del proyecto PROGENES, y su disponibilidad a lo largo de todo este tiempo. Quiero agradecer también la hospitalidad de Jean Michel Bazin, su interés por nuestro trabajo, y las edificantes discusiones que he tenido la ocasión de mantener con él.

Debo expresar igualmente mi gratitud hacia el Instituto de Ingeniería del Conocimiento de la Universidad Autónoma de Madrid, por haber dado cabida a un proyecto de investigación como éste, y haber facilitado la infraestructura y los medios que han hecho posible la realización de este trabajo.

Gracias también a todos los compañeros del IIC (en especial a los del despacho XV) que han contribuido a hacer más agradable mi estancia en el Instituto durante estos años, y en particular a mi gran amigo y compañero Andrés Canales, que me introdujo en el mundo de la Inteligencia Artificial.

Finalmente, quiero dar las gracias a Ana por el apoyo incondicional, el afecto, la comprensión y la paciencia ilimitada que me ha dado durante estos últimos meses, y por recordarme de vez en cuando cómo piensan los matemáticos. Gracias en fin a mi familia y amigos, y a toda la gente que de una u otra manera ha hecho posible que este trabajo llegue a buen término.

Indice

Everybody agrees that, in mathematics, know-how is more important, or even much more important, than mere possesion of information.

G. Polya

Capítulo 1

Introducción

La resolución automática de problemas de Matemáticas plantea necesidades muy distintas según los objetivos que se persiga alcanzar y el tipo de problemas que se quieran tratar. El sistema PROGENES pretende llegar tan lejos como sea posible en la resolución de problemas conceptuales de Matemáticas de un nivel de enseñanza media o primer curso de universidad, utilizando mecanismos de la mayor generalidad posible, desarrollados a partir de un modelo cognitivo de la actividad involucrada en la resolución de estos problemas por un matemático [CAS92, CAS94].

Para dar una idea del tipo de problemas que PROGENES es capaz de resolver, empezamos por incluir aquí unos cuantos ejemplos, la mayoría de los cuales han sido extraídos de un conocido libro de texto de Cálculo Diferencial [SPI81]. Estos problemas se utilizarán para ilustrar los mecanismos que se describen a lo largo de los sucesivos capítulos de este trabajo.

Problema 1. *Hallar la recta que pasa por el punto* $(0,0,3)$*, es paralela al plano* $2x - 2y - 2z + 3 = 0$*, y es tangente al paraboloide* $z = 3x^2 + 3y^2 + 2xy - 2x + 2y + 1$.

Problema 2. *Dadas las funciones* $f : A \to B$*,* $g, h : B \to A$*, demostrar que* $f \circ g = I$*,* $h \circ f = I \Rightarrow h = g$.

Problema 3. *Dadas* $f : A \to B$ *y* $g : B \dashrightarrow A$*, tales que* $f \circ g = I$*, la función identidad, demostrar que* g *es inyectiva y* f *es sobreyectiva.*

Problema 4. *Sea $f(x) = x^3 + 3x^2 + 3x + 2$, y sea g una función real. Demostrar que si g es inyectiva, también lo es $f \circ g$.*

Problema 5. *Demostrar que si g es una función par entonces para cualquier función real f se tiene que $f \circ g$ es par.*

Problema 6. *Demostrar que si una función f de $\mathbb{R}$ en $\mathbb{R}$ es par y derivable en el origen, entonces $f'(0) = 0$.*

Problema 7. *Demostrar que cualquier función de $\mathbb{R}$ en $\mathbb{R}$ se puede escribir como la suma de una función par y otra impar.*

Problema 8. *Sabiendo que $x + 1 \leq f(x) \leq e^x$ para todo $x \in \mathbb{R}$, demostrar que f es continua en el origen.*

Problema 9. *Demostrar que toda función lipschitziana de $\mathbb{R}$ en $\mathbb{R}$ es continua en $\mathbb{R}$.*

Problema 10. *Demostrar que si f es continua en 0, y para todo $x \in \mathbb{R}$, $f(x+y) = f(x) + f(y)$, entonces f es continua en todo $\mathbb{R}$.*

Problema 11. *Sea $f(x) = xg(x)$ para todo $x \in \mathbb{R}$. Sabiendo que g es continua en 0, demostrar que f es derivable en 0.*

Problema 12. *Sea $f : [0,1] \to [0,1]$ una función derivable en $[0,1]$ tal que para todo $x \in [0,1]$, $f'(x) \neq 1$. Demostrar que existe exactamente un x en $[0,1]$ tal que $f(x) = x$.*

Problema 13. *Hallar p y q sabiendo que $x = 1$ es un mínimo de $f(x) = x^2 + px + q$, en el que f toma el valor 3.*

Problema 14. *Sea $f : \mathbb{R} \to \mathbb{R}$ tal que f' es periódica de periodo p. Demostrar que f es también periódica de periodo p si y sólo si $f(p) = f(0)$.*

Desde sus inicios, el trabajo relacionado con el desarrollo de programas capaces de resolver problemas en Matemáticas ha estado muy arraigado en el campo de la lógica. Sin embargo, hace falta algo más que lógica para resolver problemas como, por ejemplo, los que uno puede encontrar en un curso de Matemáticas de primer año de universidad. La lógica es una herramienta muy adecuada para manejar los axiomas elementales del dominio, pero en la práctica, para tratar problemas como los enunciados más arriba, un matemático o un estudiante normalmente se despega del nivel axiomático, y utiliza el conocimiento adquirido a través de su experiencia en resolver problemas. Este conocimiento no sólo incluye definiciones y teoremas, sino también conocimiento práctico o *know-how*. No basta con saber razonar correctamente de acuerdo con las reglas de la lógica, sino que es importante también utilizar la intuición y el sentido común.

Por otra parte, el tipo de mecanismos deductivos necesarios en la clase de problemas que queremos resolver varía desde la aplicación estricta de reglas de inferencia de la lógica como el *modus ponens*, hasta la aplicación sistemática de procedimientos específicos para problemas tipo, como el estudio del comportamiento global de una función real, o la realización de cálculos formales, como la resolución de ecuaciones o el cálculo de límites, derivadas, e integrales.

1.1 Ejemplo preliminar

Para dar una primera idea de los principios generales del sistema PROGENES, consideraremos uno de los ejemplos anteriores, que nos servirá para ofrecer una visión global del funcionamiento del sistema. Empezaremos por analizar la resolución natural del problema, para después mostrar la demostración que realiza PROGENES.

Problema 6. *Demostrar que si una función f es par y derivable en el origen, entonces $f'(0) = 0$.*

1.1.1 La solución del matemático

Veamos la solución propuesta por un matemático:

Si f es par, entonces para todo $x \in \mathbb{R}$ se cumple $f(x) = f(-x)$.
Derivando en los dos miembros de esta igualdad, obtenemos $f'(x) = -f'(-x)$.
En particular, en $x = 0$, $f'(0) = -f'(0)$, lo cual nos conduce a $f'(0) = 0$.

La solución es corta, pero no trivial. Analicémosla detenidamente paso a paso. Primero se substituye el concepto de *función par* por su definición, cosa que un matemático no duda en hacer, ya que sabe que no se trata de un concepto profundo sobre el cual tengamos teoremas que permitan razonar en términos del mismo, y además la definición no introduce elementos de particular complejidad. En cambio con el concepto de derivada ocurre todo lo contrario, de ahí que su definición no se llega a aplicar en ningún momento en este problema.

Después se derivan los dos miembros de la igualdad $f(x) = f(-x)$ respecto de x. Este es el punto clave de la demostración. La cuestión es por qué se decide hacer esta manipulación, cuando son posibles otras como por ejemplo, particularizar la igualdad en $x = 0$. También podríamos haber intentado demostrar que $x = 0$ es un extremo local de f. Sin embargo, hay una razón por la cual merece más la pena intentar la opción elegida antes que otras, y es la siguiente: tenemos una hipótesis que es una igualdad, en la que f aparece *casi* despejada, teniendo en cuenta que f y $f(x)$ son *casi* lo mismo. Y por otro lado el objetivo a demostrar es también una igualdad, en cuyo primer miembro aparece también f, pero afectado por dos operadores: la *derivada* y la *toma de imagen* (en 0). Se nos puede por tanto ocurrir aplicar estos mismos operadores y en el mismo orden a los dos miembros de la hipótesis, derivando primero y particularizando en 0 después, ya que con ello conseguimos una igualdad que se acerca a la que queremos demostrar, puesto que por la forma en que se ha construido, uno de sus miembros va a ser semejante (en este caso igual) a uno de los miembros del objetivo. De modo que obtenemos:

$$\frac{d}{dx}(f(x))|_{x=0} = \frac{d}{dx}(f(-x))|_{x=0}$$

Para completar este paso, se necesita hacer un pequeño cálculo de derivadas que incluye la aplicación de la regla de la cadena. Asímismo, al substituir x por 0, aparece la expresión -0, que debemos simplificar a 0. Estas pequeñas manipulaciones son triviales desde el punto de vista de una persona con una formación básica en Matemáticas, y por ello ni siquiera se mencionan en la demostración. Sin embargo, para la mayoría de los demostradores de teoremas desarrollados hasta la fecha, es necesario construir toda una demostración para resolver este tipo de cálculos, lo cual resulta artificioso e ineficiente.

Sería deseable que el sistema emplease unos medios y un esfuerzo semejantes a los que en principio el problema requiere, generando soluciones cuya complicación no sobrepase excesivamente la de las que obtienen las personas.

Por último, el paso de $f'(0) = -f'(0)$ a $f'(0) = 0$ también se obvia en la demostración, pero requiere unas nociones elementales de manipulación algebraica, en este caso las necesarias para despejar una expresión en una igualdad.

1.1.2 La solución del sistema

Veamos ahora cómo resuelve PROGENES el mismo problema. Durante la resolución, PROGENES representa el estado del problema por medio de una base de datos, donde se almacenan los elementos del mismo. Los símbolos del problema que representan objetos dados, como f, se renombran con un subíndice, para evitar conflictos de nombres.

OBJETIVO : f_1 *derivable en* $0 \wedge f_1$ *par* $\Longrightarrow f_1'(0) = 0$

HECHOS : (1) $f_1 : \mathbb{R} \rightarrow \mathbb{R}$

Para demostrar una implicación, igual que un matemático, PROGENES supone la parte izquierda y demuestra la parte derecha:

OBJETIVO : $f_1'(0) = 0$

HECHOS :
...
(2) f_1 *derivable en* 0
(3) f_1 *par*

Para demostrar cualquier propiedad de la derivada de una función en un punto, primero debemos asegurarnos de que la función sea derivable en ese punto. En este caso, el sistema comprueba que f_1 sea derivable en 0, información que encuentra inmediatamente entre las hipótesis. Esto forma parte de un mecanismo más general según el cual la definición de los conceptos va acompañada de determinadas condiciones que son necesarias para que el concepto tenga sentido, condiciones que el sistema comprueba (demuestra, si es necesario) automáticamente. En este caso, la derivada de una función en un punto tiene sentido si la función es derivable en el punto.

En PROGENES, la definición de determinados conceptos se codifica de forma procedural como una regla de reescritura que substituye el concepto por su definición. Algunas reglas, como la correspondiente a la definición de función par, actúan de inmediato, mientras que otras, como la de derivada, no lo hacen hasta después de intentar otros métodos. En este caso, el sistema aplica la definición de función par, substituyendo (3) por:

(3') $\forall x \in \mathcal{D}_{f_1},\ f_1(x) = f_1(-x)$

donde $\mathcal{D}_{f_1}$ denota el dominio de f_1.

Mientras que la diferencia entre f y $f(x)$ es una cuestión de matiz para un matemático, en un ordenador la ambigüedad debe resolverse explícitamente. El hecho de que f y $f(x)$ es *casi* lo mismo se traduce en que cada vez que se tiene una igualdad de la forma $f(x) = expr$ para todo x del dominio de f (igualdad entre números reales), el sistema añade la correspondiente igualdad entre funciones $f = funcion\_real\,(x, expr)$. En este caso:

(4) $f_1 = funcion\_real\,(x, f_1(-x))$

Ahora PROGENES aplica un razonamiento análogo al descrito en el apartado anterior. Una metaregla del sistema se da cuenta de que f_1 aparece despejada en (4), y que aparece también en el primer miembro del objetivo, bajo derivación y evaluación en 0, de forma que decide aplicar estas dos mismas operaciones en los dos miembros de (4):

(5) $f_1'(0) = (funcion\_real\,(x, f_1(-x)))'(0)$

Obsérvese que esto es equivalente a igualar el primer miembro del objetivo a la expresión que resulta de substituir f por $funcion\_real\,(x, f_1(-x))$ en dicho miembro.

El segundo miembro de (5) es una expresión en la que una persona efectuaría de una forma directa las operaciones que aparecen indicadas, es decir, reescribiría mecánicamente la expresión, sin hacer ningún tipo de razonamiento matemático. Por supuesto que la particularización de una función en un punto o el cálculo de derivadas se apoyan en axiomas y teoremas, pero las operaciones se efectúan sin pensar en ellos. PROGENES hace algo parecido. Veámoslo paso a paso.

El sistema empieza por aplicar automáticamente ciertas propiedades procedurales asociadas al concepto de *derivada*. Se trata de rutinas de cálculo que realizan la

reescritura:

$$(funcion\_real(x, f_1(-x)))' \longrightarrow funcion\_real(x, -f_1'(-x))$$

El concepto de imagen de una función en un punto también tiene una definición procedural de aplicación inmediata, que en este caso da lugar a:

$$funcion\_real(x, -f_1'(-x))(0) \longrightarrow -f_1'(-0)$$

Otras rutinas de simplificación algebraica permiten reescribir -0 a 0. Como consecuencia de todo esto, el segundo miembro de (5) se reescribe como $-f'(0)$, y (5) se convierte finalmente en:

(5') $f_1'(0) = -f_1'(0)$

Otra metaregla detecta que (5') es una ecuación respecto de $f_1'(0)$, que es un miembro del objetivo, y por tanto considera interesante resolverla, de forma que, utilizando rutinas de resolución de ecuaciones, se obtiene la solución $f_1'(0) \rightarrow 0$, que se substituye en todo el problema, y en particular en el objetivo, que así resulta $0 = 0$, y se reescribe automáticamente (proceduralmente) a `True`, de forma que la demostración termina.

□

Obsérvese que en lugar de derivar los dos miembros de la igualdad (4) y particularizar en 0, se nos podría haber ocurrido utilizar esta igualdad para efectuar una substitución en el objetivo, pero en tal caso, habríamos obtenido como objetivo $-f'(0) = 0$, que es equivalente al que se tenía, con lo cual no habríamos avanzado nada. Así pues, el mecanismo utilizado permite obtener resultados como éste, que no se pueden alcanzar por substitución de igualdades.

Este ejemplo reúne varias de las dificultades a las que PROGENES pretende responder. En primer lugar, el problema, como tantos otros problemas, requiere capacidad para hacer inferencias, como deducir una nueva igualdad aplicando una misma transformación a los dos miembros de otra, pero también capacidad para hacer cálculos formales, como computar derivadas o resolver ecuaciones. Igual que un matemático, PROGENES obtiene de forma directa (procedural) los resultados que sabemos de antemano cómo obtener, y reserva los mecanismos de tipo deductivo para los problemas

que realmente requieren una búsqueda. Esto permite ahorrar tiempo, y además resulta más natural que resolver este tipo de cuentas aplicando explícitamente teoremas.

Por otra parte, PROGENES determina la conveniencia de derivar en una igualdad, o de despejar un término, en función del objetivo a demostrar. Esto va más allá de la mera aplicación ciega de propiedades del dominio, ya que se está razonando sobre la propia resolución del problema. El metarazonamiento ayuda a dar pasos en la buena dirección, evitando hacer intentos infructuosos.

Por último, el sistema utiliza la definición de un concepto como el de función par en una forma muy determinada, es decir, para reescribir automáticamente un hecho de la forma *f par* por su definición (nunca lo contrario). Esto lleva implícita una elección heurística, ya que el conocimiento de este concepto se podría haber representado por ejemplo mediante una regla de producción clásica: *SI f par ENTONCES* $\forall x \in \mathcal{D}_f,\ f(x) = f(-x)$, o incluso como una doble implicación que pudiera aplicarse en ambas direcciones.

1.2 Descripción somera del sistema

PROGENES utiliza una serie de bases de conocimiento relativas a diferentes áreas de Matemáticas, e integra un motor de inferencia, que proporciona las capacidades deductivas necesarias para el razonamiento conceptual, y un sistema de reescritura, que incluye la capacidad de realizar manipulaciones simbólicas procedurales, lo cual permite efectuar cómputos matemáticos directos, así como tratar determinadas expresiones conceptuales que pueden ser manejadas de forma similar [CAS92, CAS91, CAS92b].

En PROGENES, el metaconocimiento juega un papel fundamental en aspectos como la selección del conocimiento relevante para un problema, el control de la aplicación de las propiedades del dominio, la elección de la forma en que se opera con un conocimiento dado, o la utilización de heurísticas derivadas del *know-how* de un matemático [CAS93c].

El usuario puede desarrollar bases de conocimiento que reflejan una semántica profunda, distinguiendo entre *conocimiento compilado*, que básicamente corresponde a funciones y reglas de reescritura, *conocimiento deductivo*, como los teoremas enunciados en forma de implicaciones, que debe ser manipulado por el motor de inferencia, y *conocimiento declarativo*, como las definiciones sintácticas de objetos y otros conceptos. Esto enriquece las posibilidades de utilizar metaconocimiento, al no vernos forzados a

utilizar una representación exclusivamente declarativa, y de este modo se pueden desarrollar heurísticas potentes, como se verá en los ejemplos que se muestran en este trabajo.

PROGENES utiliza un lenguaje formal, basado en una jerarquía de tipos de objetos, que permite la representación de enunciados, así como la descripción del estado de la resolución de los problemas [CAS94]. El lenguaje incluye diferentes clases de conceptos, entre los que podemos distinguir *objetos*, *metafunciones* y *metapredicados*. Mientras que los primeros son entidades pasivas, estáticas, como rectas, conjuntos, números reales o vectores, los otros dos toman objetos como argumentos, y devuelven otro objeto como valor. En el caso de los metapredicados, el valor devuelto es un booleano. La intersección de dos conjuntos, la pendiente de una recta, la suma de números, o la derivada de una función, son ejemplos de metafunciones, mientras que propiedades como continuidad de funciones, ortogonalidad de vectores, o tangencia de dos superficies, son metapredicados. Tanto las metafunciones como los metapredicados tienen una definición procedural asociada que puede ejecutarse en un momento dado.

PROGENES ha sido desarrollado con el propósito de permanecer tan cerca como sea posible de la representación del conocimiento y los mecanismos de resolución de una persona. De este modo, las soluciones generadas por el sistema son naturales y fáciles de entender. Esto hace que nuestro trabajo sea particularmente interesante para aplicaciones como las relacionadas con la educación [BAZ93, CAS93b], dado que contiene el mismo cuerpo de conocimiento que un estudiante tiene que asimilar.

PROGENES es capaz de resolver muchos problemas del tipo de los que uno puede encontrar en un libro de texto, que otros demostradores de teoremas no pueden resolver, pero no es capaz en cambio de demostrar los teoremas difíciles que demuestran otros sistemas.

El contenido de este trabajo se complementa con la tesis doctoral *Fundamentación de un sistema en resolución automática de problemas*, de F. Saiz [SAI94]. Ambas son el resultado del trabajo de equipo en el proyecto PROGENES, coordinado por Roberto Moriyón, y desarrollado en el Instituto de Ingeniería del Conocimiento, y aunque cada una de ellas trata en profundidad aspectos distintos del sistema, ambas comparten inevitablemente áreas comunes. El sistema PROGENES se encuentra en la actualidad en su fase final de desarrollo, por lo que algunas de las técnicas que se describen aquí no están aún plenamente implementadas, pero están diseñadas a un nivel de detalle suficiente como para que completar su implementación no plantee problema alguno.

El resto de este trabajo esta organizado como sigue. En el siguiente capítulo se da una descripción somera del estado del arte. En el capítulo 3 se describen los elementos y mecanismos básicos del sistema, incluyendo la base de conocimiento, el lenguaje formal, la utilización de conocimiento procedural, y el razonamiento basado en reglas. Las diversas formas de metaconocimiento utilizadas por PROGENES se describen en el capítulo 4, a lo largo del cual se muestra la resolución de diversos problemas por el sistema. El capítulo 5 está dedicado a la discusión de las posibilidades del sistema como base para el desarrollo de un tutor inteligente. Finalmente, las conclusiones de este trabajo, junto con algunas perspectivas de continuación del mismo se apuntan en el capítulo 6.

Capítulo 2

Antecedentes

La idea de implementar en un ordenador la capacidad deductiva humana procede de los primeros tiempos de la Inteligencia Artificial (IA). Una de las aplicaciones típicas de la Deducción Automática es la resolución de problemas de Matemáticas y otros campos afines. Los primeros esfuerzos en este área partieron de una perspectiva muy cercana al campo de la lógica formal. El lenguaje y los mecanismos de la lógica encajan bien en los esquemas de un ordenador, y por otra parte las Matemáticas se sustentan en definitiva sobre una lógica y unos axiomas. En un principio se trabajó en el desarrollo de sistemas totalmente generales, independientes del dominio a tratar, y capaces en teoría de resolver cualquier problema [ROB65]. Aunque la capacidad teórica de estos demostradores era muy grande, los resultados que se podían obtener de esta manera eran muy limitados en la práctica, ya que el espacio de búsqueda resultaba excesivamente amplio, de modo que en gran parte del trabajo dedicado a este área se abandonó la ambición de universalidad en favor de sistemas específicos a un dominio que, aunque no son capaces de demostrar *cualquier* problema, sí permiten obtener algunos resultados en un tiempo razonable. Desde entonces hasta nuestros días, muchos investigadores se han dedicado a mejorar y perfeccionar las técnicas desarrolladas, diseñando nuevas heurísticas y construyendo modelos formales cada vez más sofisticados [WOS65, CHA70]. Otros renunciaron a la idea de construir demostradores totalmente automáticos, y trabajaron en el desarrollo de sistemas interactivos capaces de resolver problemas con la ayuda externa de una persona [MCA89], o programas de adición y verificación de demostraciones creadas por el usuario [BOY88].

Existe una gran distancia entre lo que uno puede esperar hacer utilizando lógica pura, y las Matemáticas que conocemos hoy día. En el empeño por acortar esta dis-

tancia, surge la idea de partir de la posición contraria, la del matemático, y buscar el acercamiento al terreno de la lógica, a base de formalizar los mecanismos del pensamiento matemático para adaptarlos al ordenador [BLE77]. Las ideas basadas en la cognición humana han sido un punto de referencia permanente de gran parte del trabajo realizado en este campo [PAS89]. Ambas aproximaciones, la lógica y la cognitiva, son complementarias, y muchos de los sistemas que se han desarrollado integran ambas tendencias.

Por otra parte, las áreas de las Matemáticas que tienen una fuerte componente de cálculo formal y admiten un tratamiento básicamente procedural, como la trigonometría, la integración o el álgebra lineal, han sido tratadas con éxito por sistemas de manipulación simbólica [WOL91], siguiendo una aproximación totalmente distinta de la de los demostradores de teoremas. Estos sistemas no buscan tanto reproducir en el ordenador un comportamiento inteligente, como conseguir que se resuelva el mayor número de problemas posible y de la manera más rápida posible, sin ningún tipo de compromiso respecto a las técnicas que emplean.

En este capítulo se presentarán algunos de los resultados más significativos, desde nuestra perspectiva, relacionados con estas áreas, y se dará una breve descripción de los sistemas que han tenido una mayor influencia en nuestro trabajo.

2.1 La lógica

Cualquier sistema de demostración de teoremas tiene necesariamente una vinculación con la lógica a un nivel u otro, ya que ésta es la base del razonamiento matemático. Sin embargo, una buena parte del trabajo en este campo ha hecho especial hincapié en el aspecto formal de los mecanismos de deducción, centrándose en el desarrollo de modelos teóricos sofisticados a nivel de la lógica, para después llevarlos a la práctica.

2.1.1 Principio de resolución

El *principio de resolución* [ROB65] marcó un antes y un después en el campo de la Demostración Automática de Teoremas. Este principio está basado en la demostración por reducción al absurdo, y utiliza una sola regla de inferencia, que es completa en refutación para el cálculo de predicados de primer orden. Consiste en poner los axiomas,

junto con la negación del teorema a demostrar, en forma normal conjuntiva (como conjunción de cláusulas), tras lo cual las cláusulas se combinan unas con otras derivando nuevas cláusulas, hasta obtener la cláusula vacía (es decir, *Falso*).

El principio de resolución supuso una revolución en el campo de la Demostración Automática, y dio lugar a la aparición de multitud de demostradores basados en este algoritmo. Uno de los resultados más notables del trabajo en esta dirección fue el lenguaje de programación lógica PROLOG [ROU75]. Sin embargo, tras unos primeros resultados esperanzadores, esta aproximación mostró tener grandes limitaciones en la práctica, debido a la explosión combinatoria, que hace que un resultado, aunque teóricamente obtenible, pueda no ser alcanzado en un tiempo razonable. La utilización de estrategias de control permitió mejorar los resultados [WOS65, CHA70], aunque sin alcanzar un avance definitivo. El empeño por utilizar mecanismos de inferencia completos suponía un obstáculo para la incorporación de mecanismos de control que guiasen las demostraciones en el espacio de búsqueda.

2.1.2 Demostrador de Boyer y Moore

Boyer y Moore [BOY88, BOY79] desarrollaron un sistema de demostración automática de propósito bastante general, que ha sido utilizado para demostrar teoremas en teoría de números, teoría de funciones recursivas, y lógica formal, así como en la verificación de sistemas de hardware y software. Este demostrador utiliza un paradigma de evaluación basado en lógica proposicional con igualdad y funciones. Una de sus capacidades más destacables reside en la habilidad para realizar demostraciones por inducción matemática. El sistema está fuertemente arraigado en los principios de la lógica, y su funcionamiento queda lejos del modo en que un matemático demuestra teoremas. Sin embargo, junto al conocimiento del dominio, permite utilizar conocimiento heurístico que ayuda a la aplicación pragmática y efectiva del conocimiento.

Uno de los aspectos de mayor interés en este demostrador desde la perspectiva de nuestro trabajo es la formulación procedural de los principios de inferencia lógica. Al contrario que muchos demostradores, en los que el manejo de funciones se reduce al de predicados que definen funciones implícitamente, en el sistema de Boyer y Moore la lógica proposicional se convierte en manipulación de funciones e igualdades, de forma que los mecanismos de demostración se integran en un sistema de evaluación recursiva.

El sistema parte de una serie de definiciones de funciones y tipos de datos, junto con

un conjunto de axiomas, además de los cuales puede utilizar teoremas proporcionados por el usuario, a medida que los va demostrando. Las igualdades y equivalencias lógicas que establecen estos teoremas pueden ser aplicadas como reglas de reescritura, si el usuario así lo establece por medio de una etiqueta que acompaña al teorema. La forma en que se enuncia un teorema, es decir, el aspecto sintáctico del mismo, influye también en la manera en que se realiza la reescritura. Así por ejemplo, un lema de la forma $P \Rightarrow t_1 = t_2$ da lugar a una regla de reescritura que substituye instancias[1] de t_1 por las correspondientes instancias de t_2 siempre que se cumpla P, mientras que si el lema se enuncia como $t_1 \neq t_2 \Rightarrow \neg P$, entonces la correspondiente regla substituye P por *Falso* cuando $t_1 \neq t_2$. De acuerdo con esta idea, el sistema no efectúa una normalización total de las fórmulas, sino que respeta en lo posible la forma en que han sido enunciadas, para así utilizar la información heurística implícita en su sintaxis.

El usuario puede introducir nuevos axiomas, así como definir nuevas funciones y tipos de datos. Una función f se define mediante una lista de argumentos $(a_1, \ldots, a_n)$, y un cuerpo C, de forma que la definición de f establece implícitamente la igualdad $\forall x_1, \ldots, x_n, f(x_1, \ldots, x_n) = C_{[x_1/a_1, \ldots, x_n/a_n]}$. Sin embargo, en lugar de esta igualdad, la definición de funciones se utiliza para substituir *siempre* la llamada a la función por el cuerpo de la misma (substituyendo en él los argumentos que figuran en la definición por los valores de la llamada), y la substitución inversa nunca se realiza. Así pues, la definición de una función implica una elección heurística del modo en que se va a utilizar el conocimiento que expresa.

Las funciones se reescriben sistemáticamente por su definición si no son explícitamente recursivas, es decir si no contienen llamadas a ellas mismas en el cuerpo de su definición. Las que sí lo son, en cambio, se aplican en función del resultado al que conducen, buscando obtener valores explícitos (expresiones en las que no hay nada por determinar) o expresiones que aparecen en el objetivo a demostrar, tratando de evitar una recursión infinita.

El usuario puede asimismo indicar la conveniencia de eliminar determinadas funciones, designándolas como lo que Boyer y Moore denominan *destructores*, que son símbolos que en general interesa eliminar de las expresiones (de acuerdo con el criterio del usuario). Para ello el usuario debe etiquetar determinados lemas como reglas de eliminación. El sistema incluye una heurística expresamente dedicada a la eliminación de destructores, por medio de la aplicación de los lemas etiquetados por el usuario.

[1] Un término s es una instancia de otro término t si s es el resultado de aplicar alguna substitución de las variables libres de t.

El sistema de Boyer y Moore tiene un soporte formal axiomático muy sólido, pero en contrapartida, su funcionamiento queda un tanto alejado de las demostraciones humanas, lo que explica en parte la dificultad para orientarse por sí mismo en las demostraciones difíciles. La formalización estricta y rigurosa del conocimiento del dominio a partir del nivel axiomático choca con el modo habitual de razonar de un matemático, cuyo pensamiento no siempre es formal. A menudo el matemático utiliza resultados conocidos sin preocuparse de los teoremas que subyacen (y menos aún de los axiomas que los sustentan), y tiende a efectuar demostraciones a alto nivel.

El sistema de Boyer y Moore ha sido utilizado más para la verificación de demostraciones que para el descubrimiento automático de las mismas. El papel fundamental del usuario es guiar al demostrador mediante la selección estratégica de una secuencia gradual de lemas a demostrar, así como la formulación adecuada de los mismos. El usuario debe saber por tanto cómo demostrar los teoremas en la lógica que utiliza el sistema, y comprender la manera en que éste los interpreta como reglas. Una vez puesto en marcha el demostrador, el usuario no puede ya intervenir. El sistema tratará de demostrar uno a uno los sucesivos lemas, añadiéndolos al conjunto de conocimiento disponible una vez demostrados, hasta alcanzar el resultado que se buscaba.

2.1.3 ONTIC

ONTIC [MCA89] es un demostrador semi-automatizado que, por un lado, busca ser un modelo computacional de los procesos cognitivos humanos involucrados en la demostración de teoremas, y por otro, pretende ser un sistema efectivo de verificación interactiva de teoremas en la práctica.

Una de las características principales de este sistema es la utilización de un conjunto de *focus objects*, que son especificados por el usuario, y que se utilizan para guiar los diversos mecanismos de inferencia del sistema. De esta manera, el usuario puede hacer que el sistema se fije en los objetos que permiten resolver un problema. Una vez que el usuario especifica unos focus objects, el razonamiento del sistema se restringe a los hechos relacionados con estos objetos.

Por ejemplo, ONTIC demuestra (con la ayuda del usuario) el siguiente teorema de teoría de retículos:

Si todo subconjunto de un conjunto parcialmente ordenado P tiene una cota superior mínima, entonces todo subconjunto de P tiene también una cota inferior máxima.

Este teorema se demuestra en Matemáticas tomando un subconjunto arbitrario $S \subset P$, y demostrando que tiene una cota inferior máxima. Esto se consigue considerando el conjunto M de las cotas mínimas de S, que por hipótesis tiene una cota superior mínima z, y es fácil demostrar que z es la cota inferior máxima de S. El paso clave de la demostración es la idea de considerar el conjunto M. Para que ONTIC demuestre el teorema el usuario debe indicar al sistema que considere este conjunto, es decir, debe introducirlo como focus object.

ONTIC utiliza un lenguaje formal que incluye objetos, funciones y predicados, y permite expresar enunciados matemáticos. A cada objeto del lenguaje le corresponde un tipo τ. A las funciones y predicados se les asigna una combinación de tipos de la forma $\tau_1 \times \cdots \times \tau_n \rightarrow \tau$ y $\tau_1 \times \cdots \times \tau_n \rightarrow$ *booleano* respectivamente. En este lenguaje no hay distinción entre tipos, clases y predicados de un argumento, de forma que el hecho de que un predicado es cierto sobre un determinado objeto se demuestra comprobando que el objeto es del tipo inducido por el predicado.

2.2 El conocimiento

Como ya se indicó en la introducción, también es posible abordar el problema de la Demostración Automática desde la perspectiva de las realizaciones de los matemáticos. Con ello se pueden conseguir demostraciones más rápidas, pero también más fáciles de comprender, y por tanto más útiles para el hombre.

2.2.1 Deducción natural

A principios de los años 70, Bledsoe propone acercar los mecanismos de inferencia de los programas al comportamiento del matemático, utilizando reglas de inferencia lógicas más naturales y próximas a los mecanismos deductivos humanos [BLE71]. Bledsoe fue el gran impulsor de la deducción natural [BLE77] frente a las técnicas basadas exclusivamente en el principio de resolución. Renunciando de antemano a la completitud,

desarrolló técnicas específicas al dominio, que utilizó en la demostración de teoremas sobre límites de funciones [BLE72], mejorando substancialmente los resultados obtenidos hasta entonces mediante resolución.

Una de sus aportaciones más conocidas al campo de la demostración de teoremas fue una heurística de reescritura de fórmulas [BLE71] basada en principios de deducción natural, que respeta en gran parte la sintaxis de los enunciados, según la cual, por ejemplo, para demostrar una implicación, se asume la parte izquierda, y se demuestra la parte derecha; para demostrar una conjunción, se crean tantos subproblemas como elementos haya en la misma; y para demostrar un hecho cuantificado universalmente, se elimina la cuantificación, y se substituye la variable universal por una constante genérica. Esta forma de razonar es efectivamente muy similar a la usual en Matemáticas. Estas heurísticas han sido ampliamente utilizadas en otros sistemas como MUSCADET [PAS89] y ANALYTICA [CLA93], que presentaremos más adelante, así como en PROGENES.

2.2.2 Sistemas expertos

En la búsqueda de mejores resultados, cobró importancia la idea de tomar como modelo el comportamiento del matemático, y proporcionar al sistema una gran cantidad de *conocimiento* específico del dominio, partiendo de la premisa de que en dominios con un fuerte contenido conceptual, el ser humano es el mejor resolvedor de problemas conocido hasta el momento. De ahí la idea de utilizar bases de conocimiento y desarrollar sistemas expertos.

En los sistemas expertos, el conocimiento se representa por lo general en forma de hechos (y/u objetos) y reglas. El mecanismo básico de inferencia consiste en el razonamiento basado en reglas, mediante el cual una regla se ejecuta cuando se cumplen sus condiciones. La base de conocimiento es relativa a un dominio, y el motor de inferencia que la interpreta es general, aunque puede (y suele) estar adaptado a una clase de dominios.

Uno de los primeros y más célebres sistemas expertos fue MYCIN [SHO76, BUC84], inicialmente desarrollado para su aplicación en el campo de la medicina, y también utilizado en el dominio de las inversiones.

Con la idea de incluir conocimiento en el sistema vino la preocupación por emplear

mecanismos de control del proceso de deducción similares a los que utiliza el experto para guiar su razonamiento [DAV80]. Es decir, se trata de introducir en el sistema, junto con el conocimiento, el *metaconocimiento* que permite la gestión y utilización apropiada de éste. De aquí vino la idea de las *metareglas* [DAV80b], que son reglas que razonan en términos del estado del problema en sí, es decir, razonan sobre el *metaproblema*. Las reglas tienden a ser específicas a un dominio, mientras que las metareglas pueden ser más generales.

El sistema TEIRESIAS [DAV82] es una extensión de MYCIN que incorpora metaconocimiento en forma de metareglas que permiten modificar la base de conocimiento, así como centrar la atención del sistema sobre determinadas líneas estratégicas. Las estrategias pueden ser enunciadas explícitamente y utilizadas por las metareglas. Otra de las características de este sistema es que las metareglas hacen referencia a otras reglas por descripción, lo cual permite una mayor expresividad que otras formas de referencia, como la selección mediante el nombre.

Este tipo de técnicas han sido llevadas al campo de la resolución de problemas matemáticos, en sistemas como MECHO [BUN79] y MUSCADET [PAS89], que se describen más adelante. Los mecanismos basados en el conocimiento pretenden mejorar la eficiencia de los sistemas, al tiempo que dan lugar a un modo de funcionamiento más natural, y por tanto más fácil de comprender y utilizar por una persona.

Una de las mayores dificultades en el desarrollo de un sistema experto estriba en formalizar el razonamiento natural del experto humano, que a menudo se apoya en la intuición y el sentido común. Pese a ser las Matemáticas un dominio muy formalizado, construido sobre una axiomática bien definida, en la práctica, la manera de hacer Matemáticas de los matemáticos no es del todo formal, y no está exenta ni mucho menos de intuición y sentido común, lo cual hace necesario un gran esfuerzo de formalización de este tipo de razonamiento, para poder utilizarlo en el ordenador.

La idea de desarrollar técnicas de demostración a partir del conocimiento humano, junto con el interés por desarrollar aplicaciones relacionadas con la enseñanza, dieron entrada en este campo a teorías del ámbito de las ciencias cognitivas [AND90]. El aspecto cognitivo juega un papel fundamental en el desarrollo de demostradores con fines pedagógicos, ya que no se trata sólo de que el sistema sea capaz de demostrar muchos y muy difíciles teoremas, sino de que lo haga de forma similar a como se espera que lo haga el alumno.

2.2.3 Tutores inteligentes

La enseñanza asistida por ordenador ha sido una de las aplicaciones fundamentales de la investigación en la resolución automática de problemas. El aspecto central del trabajo en este terreno es la modelización cognitiva de los elementos que intervienen en el proceso de la enseñanza. Los tutores inteligentes se componen típicamente de tres módulos principales: el modelo del dominio, el modelo del alumno y el modelo del profesor (ver [SLE82]). Este último representa el conocimiento y metaconocimiento pedagógico del sistema, y es el que gestiona la actividad de la aplicación. Está interconectado con los otros dos módulos, y establece la comunicación con el usuario, por medio de una interfaz. El modelo del alumno permite interpretar y predecir el comportamiento del usuario, comprender lo que hace y diagnosticar sus errores.

El modelo del dominio o modelo del experto es el módulo capaz de resolver los problemas. Se trata en definitiva de un resolvedor automático integrado en un sistema interactivo. Este resolvedor debe estar basado en un modelo adaptado al alumno, es decir, debe utilizar mecanismos lo más naturales y parecidos posible a los que puede utilizar una persona, de forma que el usuario pueda comprenderlos y asimilarlos [FIN91]. Desde este punto de vista, PROGENES, dada la filosofía con la que ha sido desarrollado, puede hacer el papel de módulo experto, integrándose en un entorno educativo [BAZ93, CAS93, CAS93b].

En el área de las aplicaciones pedagógicas en Matemáticas es de destacar el trabajo de J. R. Anderson [AND90], que desarrolló una teoría cognitiva (ACT) [AND83], llevada posteriormente a la práctica en tres tutores inteligentes para la enseñanza en problemas de Geometría y Algebra, y programación en Lisp.

Uno de los aspectos más interesantes de estos tutores, desde nuestro punto de vista, es la compilación del conocimiento declarativo. El sistema, igual que una persona, adquiere el conocimiento en forma declarativa, representando los conceptos por medio de estructuras que contienen información sobre el tipo de concepto, su misión y su aspecto (sintaxis). Estas estructuras se utilizan inicialmente mediante mecanismos de interpretación, y por tanto de forma ineficiente. A través de la experiencia, se generan reglas de producción a partir de ellas, dándoles así una representación operativa y más eficiente.

Estos sistemas realizan un diagnóstico de los errores del usuario, a partir de una colección de los errores más frecuentes. El tratamiento de los errores no es sistemático,

sino que está programado caso por caso. Cuando el sistema detecta en el usuario una falta de conocimiento en un aspecto determinado, propone nuevos problemas para subsanarla, tras lo cual retorna al problema original.

Los tutores de Anderson se hacen cargo automáticamente de algunas operaciones de bajo nivel, como por ejemplo el balance de paréntesis (en el tutor de Lisp), que se consideran irrelevantes. De esta forma el usuario puede centrarse en problemas de más alto nivel, que son los que realmente interesan. Este es un aspecto que también se recoge en PROGENES: el sistema realiza de forma procedural, mediante *cajas negras*, los cálculos que se consideran rutinarios para un estudiante primer curso de universidad.

La línea que separa lo irrelevante o trivial de lo que no lo es, es evidentemente arbitraria, y responde a un *contrato didáctico* que se establece implícitamente a priori entre el tutor y el alumno. Por ejemplo, en los primeros cursos de enseñanza básica, es importante que los alumnos aprendan a sumar, restar, multiplicar y dividir, mientras que en la enseñanza media normalmente se permite que los estudiantes utilicen la calculadora. En cada caso, se supone que el estudiante está suficientemente experimentado en ciertos dominios, y no necesita por tanto perder tiempo ejercitándose en ellos.

Por último, Anderson hace un énfasis especial en la importancia de la comunicación con el usuario por medio de una interfaz. El tutor de Geometría, por ejemplo, muestra al usuario los pasos de su demostración, a medida que la va construyendo, en forma de un grafo en pantalla.

2.3 El metaconocimiento

Los matemáticos no utilizan sólo su conocimiento del dominio, sino también conocimiento sobre métodos, estrategias de resolución, etc. Al resolver problemas de cierta complejidad, suelen razonar en dos niveles: por una parte, construyen inferencias a nivel del dominio, y por otra, razonan sobre los objetivos a alcanzar, la situación del problema y el conocimiento de que disponen para resolverlo. Esta capacidad de *metarazonamiento* permite al matemático observar la resolución de los problemas al mismo tiempo que la realiza, controlando los pasos que se van tomando, y dirigiéndolos hacia estados próximos a la solución.

En la línea de aprovechar el conocimiento humano para construir mejores sistemas

y con motivo de la gran necesidad de mecanismos que guíen las demostraciones, el estudio del metaconocimiento de un experto aparece como una manera de obtener métodos de control útiles y eficaces que se pueden introducir en los demostradores. Al mismo tiempo, esta idea sirve al objetivo de desarrollar sistemas parecidos al ser humano.

2.3.1 El análisis de Pitrat

El trabajo más exhaustivo realizado hasta el momento sobre el metaconocimiento en general es el análisis que presenta J. Pitrat en su obra [PIT90] sobre el papel del metaconocimiento en la Inteligencia Artificial. Pitrat analiza en profundidad la utilización del metaconocimiento en los sistemas actuales de IA, en relación con la naturaleza y características del metaconocimiento en la actividad humana inteligente.

Pitrat distingue tres tipos de metaconocimiento: conocimiento sobre las propiedades del conocimiento, conocimiento para manipular el conocimiento, y conocimiento sobre los individuos. El primer tipo incluye información sobre el origen de los conocimientos (aspecto fundamental en los *Truth Maintenance Systems*), la plausibilidad o probabilidad de los mismos (como en los *sistemas borrosos*), la precisión (de los resultados numéricos), la clasificación en base a distintos criterios (misión del conocimiento, propiedades comunes, etc.), y el interés del conocimiento.

El conocimiento sobre los individuos se refiere a las personas o sistemas con los que un sistema debe interaccionar. Este conocimiento consiste en un modelo de tales individuos, que permite predecir al sistema el comportamiento, las posibilidades, o las necesidades de los mismos. Este aspecto es fundamental en el desarrollo de sistemas de enseñanza, como ya se ha comentado anteriormente. El sistema puede igualmente tener un modelo de sí mismo, que le permite controlar la actividad propia o la de distintos submódulos que lo componen.

El conocimiento para manipular el conocimiento interviene en la adquisición, almacenamiento, activación, utilización, descubrimiento y comunicación del conocimiento. El descubrimiento del conocimiento tiene que ver con los sistemas de aprendizaje automático, y se sale por tanto del ámbito de interés del presente trabajo. La adquisición y comunicación del conocimiento juegan un papel importante en las aplicaciones didácticas, en las que es necesaria una buena interacción con el usuario. Entre las funciones enumeradas, la activación y utilización del conocimiento son las que más nos concier-

nen en nuestro trabajo. Respecto a la primera, muchos sistemas, entre ellos PROGENES y MUSCADET, no utilizan para resolver un problema todo el conocimiento de que disponen, sino que seleccionan el conocimiento pertinente al problema por medio de indicios y palabras clave presentes en el enunciado. De esta manera se evita el coste que supone manejar un gran volumen de información innecesaria. En cuanto a la utilización del conocimiento, es importante que el sistema sepa decidir en qué momento y de qué manera es más conveniente aplicar un determinado conocimiento. Este (meta)conocimiento heurístico permite al sistema controlar la búsqueda de soluciones, de forma que éstas se alcancen lo más rápidamente posible.

Pitrat plantea asimismo un modelo de sistema capaz de compilar el conocimiento declarativo a conocimiento procedural, de forma que el usuario pueda proporcionar al sistema en forma declarativa el conocimiento relativo al dominio en que debe funcionar el sistema, así como el (meta)conocimiento para utilizar el conocimiento, e incluso el (meta)metaconocimiento necesario para compilar el metaconocimiento, y así sucesivamente. Finalmente, el sistema incluiría el mínimo (inevitable) conocimiento procedural necesario para iniciar el proceso.

2.3.2 MUSCADET

MUSCADET [PAS89, PAS84] es un sistema de demostración de teoremas que tiene muchas de las características de un sistema experto, y que ha logrado demostrar teoremas de cierta complejidad sobre Espacios Vectoriales Topológicos y Teoría de Conjuntos. En lugar de desarrollar un modelo teórico complejo para el tratamiento de la axiomática del dominio, MUSCADET parte de una aproximación empírica. Este sistema utiliza técnicas de resolución parecidas a las que aplican los matemáticos, lo cual le permite trabajar a alto nivel, utilizando tanto el conocimiento (del dominio) como el metaconocimiento (de resolución de problemas) propios de un experto para tratar problemas de dominios conceptuales difíciles. Las demostraciones así obtenidas son naturales y claras (especialmente si se comparan con las que proporcionan otros demostradores), y se expresan en un lenguaje similar al que emplean los matemáticos.

PROGENES debe muchas de sus ideas a MUSCADET, en particular en lo que se refiere a la utilización de conocimiento y metaconocimiento matemático de alto nivel, siguiendo una aproximación semejante a la de los sistemas expertos.

MUSCADET utiliza una base de conocimiento en la que se codifica el conocimiento del dominio (definiciones, teoremas, etc.), junto con estrategias de resolución y (meta)conocimiento heurístico que permite utilizar el conocimiento de una manera conveniente. Muchas de las heurísticas del sistema se han elaborado a partir de la observación directa del trabajo de los matemáticos y de la formalización del *know-how* que utilizan en sus demostraciones. La base de conocimiento contiene reglas y metareglas, que son interpretadas por medio de un motor de inferencia forward. Las metareglas son reglas que aplican metaconocimiento y actúan sobre otras reglas, creándolas, activándolas o reordenándolas. En MUSCADET las reglas y las metareglas se representan de la misma manera, y el motor de inferencia no las diferencia, al contrario que en PROGENES, donde, como veremos, existen ciertas diferencias entre ambos tipos de reglas.

Durante la resolución de los problemas, la información que describe el estado de la demostración se recoge en una base de datos con distintos items: hipótesis, objetivo, reglas activas, reglas aplicables, variables buscadas, etc. El sistema permite cierta flexibilidad para que el usuario defina su propia estructura de la base de datos. Antes de ser almacenadas en el item correspondiente, las fórmulas son preprocesadas según un mecanismo similar al algoritmo *split* de Bledsoe [BLE71]. Este tratamiento de las expresiones incluye además la eliminación de los símbolos funcionales mediante la introducción de nuevas variables y la adición de igualdades de la forma $z = f(x)$, que no son tratadas como tales, sino como si estuvieran en la forma $\mathcal{P}(z, x)$, donde $\mathcal{P}$ es un predicado (es decir, no se utilizan, por ejemplo, para realizar substituciones).

Uno de los mecanismos más interesantes de MUSCADET es su capacidad para crear nuevas reglas a partir de definiciones matemáticas de conceptos contenidas en la base de conocimiento. Las reglas así creadas permiten reemplazar los conceptos que aparecen en el objetivo por su definición. En las hipótesis, los conceptos no se substituyen por su definición, sino que ésta se añade a los hechos por medio de las reglas creadas. Así por ejemplo, a partir de la definición de la imagen de una función sobre un conjunto:

$$\text{IMAGEN}(f, A) = \{y \mid (\exists x \in A)\ y = f(x)\}$$

se construyen automáticamente las reglas:

Regla IMAGEN1 :
si $W = \text{IMAGEN}(f, A)$, $y \in W$
entonces AÑADE_HYP $(\exists x \in A)\ y = f(x)$.

Regla IMAGEN2 :
si $W = \text{IMAGEN}(f, A)$, $x \in A$, $y = f(x)$,
entonces AÑADE_HYP $y \in W$

Regla DEF_CONCLi :
si CONCL $y \in W$, $W = \text{IMAGEN}(f, A)$,
entonces NUEVA_CONCL $(\exists x \in A)\ y = f(x)$.

La primera regla establece que cuando se sabe que $y \in f(A)$, se añada el hecho de que $\exists x \in A$ *tal que* $y = f(x)$; la segunda determina que cuando $x \in A$ se infiera $f(x) \in f(A)$ (siempre que $f(A)$ exista en el problema); y por último la tercera regla es la que permite substituir un objetivo de la forma $y \in f(A)$ por su definición: $\exists x \in A$ *tal que* $y = f(x)$.

PROGENES utiliza algunas técnicas similares a las de MUSCADET: el conocimiento se representa en bases de conocimiento que contienen definiciones, junto con reglas y metareglas que son interpretadas por un motor de inferencia. El propósito de nuestro trabajo es extender y potenciar la capacidad heurística de mecanismos como los de MUSCADET mediante la incorporación de conocimiento procedural y capacidades de cálculo formal.

2.3.3 Otros trabajos

MECHO [BUN79] es un sistema implementado en PROLOG que resuelve problemas de mecánica elemental, a partir de una descripción de los conceptos del dominio y las leyes que rigen en éste. Este sistema utiliza metaconocimiento para decidir la estrategia a seguir en la resolución de los problemas. Uno de los mecanismos principales consiste en un método de extracción de ecuaciones que utiliza metaconocimiento sobre qué magnitudes pueden ser determinadas mediante qué fórmulas (igualdades que representan leyes físicas), qué tipo de magnitud queremos determinar, y qué fórmulas han dado lugar (por instanciación) a las ecuaciones actuales de la base de datos.

MECHO utiliza una representación de los conceptos basada en una jerarquía de tipos, similar a la que utiliza PROGENES. El lenguaje utilizado incluye funciones y predicados, que se aplican a objetos de tipos determinados. Los conceptos pueden tener asociadas ciertas restricciones, que pueden dar lugar a hechos que se dan por ciertos, o a condiciones que hay que comprobar, dependiendo del contexto en que aparece el concepto.

El sistema es capaz de reconocer configuraciones globales a partir de las relaciones entre los objetos del problema (cuerdas, barras, poleas, masas puntuales, etc.), reconstruyendo así la disposición de los mismos, utilizando información por defecto para completar la información que no se indica.

Aunque los objetivos que persigue son muy distintos a los nuestros, merece la pena citar asimismo el sistema AM [DAV82], concebido para el *descubrimiento* de conceptos y teoremas matemáticos. Este sistema no demuestra teoremas, sino que los *propone*. A partir de una base de conocimiento de conceptos elementales de Teoría de Conjuntos y una serie de heurísticas de exploración, AM inventa nuevos conceptos que considera *interesantes*, utilizando métodos no rigurosos.

El sistema EURISKO [LEN83] fue desarrollado a partir de AM, extendiendo los mecanismos de su predecesor para tratar otros dominios, no necesariamente matemáticos, y potenciando la utilización de metaconocimiento. Este sistema es capaz de descubrir y modificar sus propias heurísticas.

2.4 Cálculo formal

Algunas áreas de las Matemáticas como la trigonometría, la integración, o la manipulación algebraica, tienen una gran componente de cálculo formal. La mayoría de los sistemas de Demostración Automática de Teoremas carecen de la capacidad de realizar cómputos matemáticos, y desarrollan soluciones sofisticadas y artificiosas para demostrar un cómputo tan sencillo como $1 + 1 = 2$, lo cual resulta muy poco natural y en absoluto eficiente. El tipo de Matemáticas que se pueden hacer sin capacidad de cálculo es muy restringido.

En los últimos años se han desarrollado diversos programas de manipulación simbólica, como Mathematica [WOL91], que son capaces de tratar satisfactoriamente una gama muy amplia de problemas de este tipo, con una aproximación totalmente distinta

a la de la Demostración Automática de Teoremas. Estos sistemas son útiles en dominios que admiten un tratamiento esencialmente procedural, es decir, para los cuales es posible determinar algoritmos bien definidos que resuelven determinadas clases de problemas.

Mathematica, por ejemplo, dispone, entre otras, de rutinas para el cálculo de límites, derivadas, integrales, series, manipulación algebraica, resolución de ecuaciones numéricas y diferenciales, y álgebra lineal. Este sistema ofrece además un entorno de programación que permite al usuario combinar operaciones para realizar manipulaciones complejas. Uno de los mecanismos más potentes de Mathematica es su sistema de reescritura y evaluación simbólica. El usuario puede definir sus propias reglas de reescritura de la forma $lhs := rhs$, estableciendo que una expresión que haga *pattern-matching* con lhs debe ser automáticamente substituida allí donde aparezca por la instanciación correspondiente de rhs. El programa utiliza un paradigma de evaluación recursiva, según el cual las expresiones se reescriben hasta que se obtiene un resultado al que no se puede aplicar ninguna regla.

Sin embargo, la mayoría de los campos de las Matemáticas tienen ambas componentes, la conceptual y la simbólica, y requieren capacidad para *razonar*, y capacidad para *calcular*. Como ya apuntó Bledsoe [BLE72],

> *Mathematicians usually leave to calculation those things that can be calculated, thereby freeing their mind for the harder task of finding inferences.*

2.4.1 ANALYTICA

Clarke y Zhao [CLA93] desarrollaron recientemente el sistema ANALYTICA, implementado sobre Mathematica y con carácter experimental, que añade técnicas deductivas basadas en el cálculo de *sequents* a la capacidad de manipulación simbólica de Mathematica. ANALYTICA es capaz de resolver problemas que se pueden expresar en términos de manipulaciones algebraicas, incluyendo desigualdades y sumas infinitas. Además de los mecanismos deductivos generales, ANALYTICA utiliza heurísticas específicas para el tipo de problemas que trata. El sistema es capaz de demostrar, por ejemplo, las propiedades de la proyección estereográfica, o estudiar la diferenciabilidad de una función definida mediante una suma infinita.

Las capacidades de razonamiento conceptual del sistema son sin embargo muy lim-

itadas. Los conceptos que aparecen en los problemas se substituyen inmediatamente por su definición, de forma que el sistema trabaja siempre a nivel de expresiones algebraicas. En este sentido, ANALYTICA es más una extensión de Mathematica mediante mecanismos de deducción, que un demostrador que integre a Mathematica como módulo de cálculo formal.

2.5 Conclusión

¿Dónde se sitúa PROGENES en este panorama? Nuestro sistema trabaja sobre un tipo de problemas de un nivel semejante al de MUSCADET, es decir, problemas conceptuales que requieren un conocimiento profundo, en dominios de alto nivel en el sentido de que están muy por encima de la axiomática del dominio. No parece muy viable en la práctica resolver problemas sobre continuidad de funciones pensando en los axiomas de los números reales, y desde luego, no es así como lo hacen los matemáticos. Es por tanto muy válida para nuestros objetivos la perspectiva de MUSCADET de utilizar el conocimiento *real* de un experto (y no sólo el formal), y desarrollar heurísticas de control a partir del metaconocimiento de los matemáticos.

Sin embargo, los matemáticos también tienen la capacidad de hacer cálculos. PROGENES incluye igualmente capacidades de este tipo, lo que le permite superar las limitaciones de otros sistemas a la hora de hacer cálculos que no representan ninguna dificultad para una persona. Esta capacidad se incluye dentro del contexto de la utilización de conocimiento procedural, que permite acelerar y dirigir (de antemano) la resolución de los problemas. En este sentido, PROGENES tiene puntos en común con el demostrador de Boyer y Moore. Al tipo de mecanismos de este sistema, PROGENES añade la capacidad de guiar las deducciones mediante metaconocimiento. Al mismo tiempo, los mecanismos procedurales de PROGENES aportan mayores posibilidades para expresar y utilizar metaconocimiento.

Por otra parte, PROGENES representa un modelo cognitivo de un experto en el dominio, y como tal tiene un interés didáctico evidente, dado que el sistema refleja una estructura de conocimiento similar a la que debe adquirir un estudiante. En este contexto, PROGENES puede ser integrado en un entorno interactivo de enseñanza, extendiendo el resolvedor con funciones pedagógicas como las descritas en el apartado 2.2.3.

Capítulo 3

Descripción del sistema

El nivel de efectividad de un resolvedor de problemas depende en gran medida de la elección de una representación adecuada del conocimiento. A este respecto, el modelo humano ha sido una referencia permanente en la construcción de bases de conocimiento para PROGENES, lo cual ha permitido la implementación de heurísticas *naturales* basadas en un modelo del experto. Como consecuencia, la representación interna del dominio resulta fácil de entender para una persona, puesto que se basa en unos esquemas similares a los suyos. Esto facilita por un lado el trabajo de desarrollo, y por otro hace que los resultados obtenidos puedan tener interés de cara a aplicaciones educativas [CAS93b, CAS93].

PROGENES utiliza una serie de **bases de conocimiento** relacionadas con diferentes áreas de Matemáticas, que contienen por un lado una descripción de los conceptos del dominio, incluyendo tanto aspectos declarativos como procedurales, y por otro una serie de reglas y metareglas en las que se recogen teoremas o propiedades sobre los conceptos. Las metareglas representan estrategias de control que pueden ser generales o específicas a una parte del dominio. El metaconocimiento que se utiliza para controlar la aplicación de un teorema puede ir unido a éste en una misma metaregla que incluye a ambos, como veremos en capítulos posteriores.

Las definiciones de las bases de conocimiento determinan un **lenguaje formal** basado en una **jerarquía de tipos**, que permite representar enunciados, así como los distintos objetos y elementos necesarios para describir el estado de la resolución de un problema. Este lenguaje puede cumplir dos funciones: **descriptiva**, como mero vehículo de representación *declarativa* de elementos de un problema, y **operacional**,

por la capacidad *deductiva* de reglas y metareglas, junto con la semántica *procedural* que acompaña a algunos de los conceptos que conforman el lenguaje, y que permite de forma directa avanzar en la resolución de un problema.

Durante la resolución de un problema, PROGENES utiliza una **base de datos** en la que se refleja el estado en que se encuentra la resolución de un problema mediante distintos *items*: hechos, reglas, objetivo, objetos buscados, objetos conocidos, etc.

Por otra parte, el sistema integra un **motor de inferencia** y un **sistema de evaluación y reglas de reescritura**, mecanismos que permiten aplicar el conocimiento contenido en las bases. El motor de inferencia gestiona la parte deductiva del sistema, que consiste en la aplicación de reglas y metareglas a la base de datos, entendiendo las expresiones contenidas en ésta como información declarativa. La interpretación procedural del lenguaje formal PROGENES se lleva a cabo mediante evaluación, reescritura y llamadas a rutinas predefinidas, gracias a lo cual es posible aplicar de forma directa ciertas definiciones o propiedades, precisar métodos para realizar tareas que se pueden descomponer en una secuencia de pasos bien conocida, o ejecutar procedimientos de cálculo.

PROGENES incluye un módulo de cálculo simbólico y combina la capacidad de manipulación simbólica de este módulo, que permite resolver de forma procedural diversos (sub)problemas de cálculo formal, con las capacidades deductivas del motor de inferencia. De esta forma, el conocimiento declarativo (axiomas, objetos, etc.), puede interpretarse en un momento dado como expresión de conocimiento procedural (llamadas a funciones, reglas de reescritura, etc.). Como consecuencia, PROGENES es capaz de resolver fácilmente problemas que involucran tanto aspectos conceptuales como computacionales.

Una de las estrategias principales del sistema es el razonamiento basado en el objetivo, que en muchas ocasiones se utiliza en conjunción con el Algoritmo de Expansión, que reformula problemas sobre conceptos complejos (puntos, superficies, sistemas mecánicos, etc.) en términos de otros más simples que el módulo de cálculo formal puede manejar (números, funciones, ecuaciones diferenciales, etc.). El algoritmo de expansión se describe con detalle en [CAS94, SAI94], y no será descrito en esta tesis.

Este capítulo está dedicado a la representación del conocimiento por una parte, y por otra a la descripción de los principales mecanismos de resolución. El primer aspecto se desarrolla en las secciones 3.1 a 3.4. En la primera sección se presentan una serie de

ejemplos, que sirven como pretexto para introducir informalmente las nociones que se describen después con detalle. La sección 3.2 contiene la descripción de los elementos que conforman una base de conocimiento. Las dos secciones siguientes se ocupan del lenguaje PROGENES y los mecanismos orientados a objetos respectivamente. En la sección 3.5 se describe la componente procedural del sistema, y los mecanismos de tipo deductivo se estudian en la 3.6. Por último, tras una breve recapitulación del conjunto de los mecanismos descritos (sección 3.7), en las dos últimas secciones del capítulo se presenta la resolución de dos problemas, a modo de ilustración de todo lo visto a lo largo del capítulo.

Notación

PROGENES representa los enunciados y las distintas expresiones matemáticas utilizando un lenguaje formal propio. Como veremos en la sección 3.3, las expresiones PROGENES se forman siempre mediante un conector o constructor principal seguido de una lista de componentes o argumentos, en notación prefija. Sin embargo, para facilitar la comprensión de los ejemplos que se presentan en este trabajo, utilizaremos una notación equivalente pero más cercana a la simbología matemática.

Esta notación hace uso de diversas abreviaturas, como $P \wedge Q$ en lugar de `and(`P`,` Q`)`, $\exists x \in \mathbb{R}\ P$ por `existe(`x`:real,`P`)`, $A \cap B$ en lugar de $\cap(A,B)$, $x \in A$ por $\in(x, A)$, ó $\{x \in \mathbb{R} \mid P(x)\}$ para denotar `conjunto(`x`:real,`$P(x)$`)`. Asimismo, $(x-1)^2 + y^2 + z^2 = 4$ es una abreviatura de `=(+(exp(-(`x`,1),2), exp(`y`,2), exp(`z`,2)), 4)`, $\lambda(x, y).\sin\frac{1}{x}$ representa `lambda(lista(`x`,`y`), sin(÷(1,`x`)))`, $f(x)$ es `imagen(`f`,`x`)`, $\mathcal{D}_f$ y $\mathcal{R}_f$ denotan `dominio(`f`)` y `rango(`f`)` respectivamente, $\lim_{x \to a} f(x)$ se utilizará en lugar de `limite(imagen(`f`,`x`), `x`, `a`)`, f' en lugar de `derivada(`f`)`, y f *inyectiva* en lugar de `inyectiva(`f`)`. También se utilizarán contracciones como $x < y < z$, que significa $x < y \wedge y < z$, ó $f : A \to B$ para denotar $\mathcal{D}_f = A \ \wedge \ \mathcal{R}_f = B$.

En general, se intentará que la notación, sin dejar de ser equivalente a la representación interna, sea lo más natural y parecida posible a la usual en Matemáticas.

3.1 Ejemplos

Para ilustrar la descripción del sistema utilizaremos tres de los ejemplos enunciados en la introducción. La resolución completa de los dos primeros se mostrará al final de este capítulo.

Problema 4. *Sea $f(x) = x^3 + 3x^2 + 3x + 2$, y sea g una función real. Demostrar que si g es inyectiva, también lo es $f \circ g$.*

Este problema requiere una base de conocimiento relativa a *conjuntos*, en las que se da la definición de los conceptos involucrados en el problema. Esta base de conocimiento incluye la definición de (a) **objetos** como `conjunto` y `funcion_real` (función de $\mathbb{R}$ en $\mathbb{R}$); (b) **metafunciones** como el `dominio` y el `rango` de una función, la `composicion` de dos funciones, y la `imagen` mediante una función de un elemento de su dominio; y (c) **metapredicados** como `inyectiva`, que se aplica a funciones, y $\in$, que opera sobre un objeto y un conjunto.

La base contiene además una lista de teoremas relativos al tipo de objetos, metafunciones y metapredicados incluidos en las mismas, como el teorema que dice que la composición de dos funciones inyectivas es inyectiva. Cada teorema está ligado a los conceptos con los que está relacionado, de forma que los teoremas que son relevantes para un problema pueden ser seleccionados al iniciarse la resolución de éste.

PROGENES incluye una serie de bases de conocimiento predefinidas, que incluyen a la mayoría de las que se muestran aquí.

Los objetos se definen en la base de conocimiento indicando la información necesaria para determinarlos de forma constructiva. Por ejemplo, una función se determina mediante dos conjuntos (su dominio y su rango), y una regla que nos permite determinar la imagen de cualquier elemento del dominio. Esta regla es una *λ-expresión* formada por expresiones del lenguaje formal que pueden ser proceduralmente computables por el sistema. Así por ejemplo, la función $f(x,y) = \frac{1}{x^2+y^2}$ se representa como:

$$\texttt{funcion}\ (\mathbb{R}^2 \setminus \{\texttt{punto(0,0)}\},\ \mathbb{R},\ \lambda(x,\ y).\frac{1}{x^2+y^2})$$

Las funciones de $\mathbb{R}$ en $\mathbb{R}$ son un caso particular del tipo `funcion`, pero sin embargo su forma de representación es distinta, ya que carecen de los slots `dominio` y `rango`, que se definen como metafunciones. Como veremos más adelante, en el lenguaje PROGENES no hay herencia de estructura, aunque sí abstracción de datos.

Las metafunciones son símbolos que permiten la construcción de objetos mediante la ejecución de código procedural. Por ejemplo, la metafunción `composicion` se puede aplicar sobre dos funciones construidas en la forma descrita en el párrafo anterior, suponiendo que el rango de la primera está contenido en el dominio de la segunda. El resultado es una nueva función cuyo dominio es el de la primera, cuyo rango es el de la segunda, y cuya regla corresponde a la de la segunda función con la substitución correspondiente. Por ejemplo, la expresión:

`composicion (funcion_real (`x`, sin`x`),`
`funcion (`$\mathbb{R}^2 \setminus$ `{punto(0,0)}, `$\mathbb{R}$`, `$\lambda(x,\ y).\frac{1}{x^2+y^2}$`))`

puede ser directamente substituida por (es decir, evalúa a):

`funcion (`$\mathbb{R}^2 \setminus$ `{punto(0,0)}, `$\mathbb{R}$`, `$\lambda(x,\ y).\sin\frac{1}{x^2+y^2}$`)`

Para ello es necesario primero *traducir* la expresión `funcion_real(`x`, sin`x`)` a la forma general de representación de funciones, es decir, `funcion(`$\mathbb{R}$`, `$\mathbb{R}$`, `$\lambda x.\sin x$`)`, ya que la metafunción `composicion` está definida a nivel del tipo `funcion`, y espera recibir sus argumentos en una forma acorde con la estructura correspondiente a este tipo.

Los tipos definidos en la base de conocimiento están organizados según una jerarquía, que va acompañada de una serie de rutinas de traducción que permiten la conversión de un objeto de un tipo dado a cualquiera de sus supertipos. Gracias a esto, se pueden definir metafunciones sobre clases generales, y aplicarlas después a cualquier subclase. De hecho, una metafunción se puede definir mediante diferentes procedimientos para diferentes clases, y en el momento de aplicarla se hace una búsqueda hacia arriba en la jerarquía, partiendo de los tipos de los argumentos, hasta que se encuentra la definición que les corresponde. Entonces los objetos sobre los que se está aplicando la metafunción se traducen a la superclase correspondiente. Este mecanismo es similar al que se utiliza habitualmente en Diseño Orientado a Objetos para la aplicación de métodos; en realidad, el formalismo usual es un caso particular del nuestro, en el que las funciones de traducción consisten en eliminar ciertos slots.

Los metapredicados asignan un valor booleano (`True` ó `False`) a uno o más objetos. De manera análoga a las metafunciones, esto se lleva a cabo mediante la ejecución de un programa asociado al predicado. En este sentido, los metapredicados se pueden ver como metafunciones cuyo valor de retorno es de tipo `booleano`. De hecho, a menudo utilizaremos la palabra *metafunción* para referirnos a ambos conceptos.

El prefijo *meta-* de las metafunciones tiene dos propósitos. Por una parte, permite distinguir entre `funcion`, como concepto del dominio matemático, que es un tipo definido en la base de conocimiento, y *metafunción* en el sentido descrito más arriba. Por otra parte, el prefijo responde al hecho de que toda metafunción, igual que todo metapredicado, lleva implícito metaconocimiento sobre cómo tratar expresiones al resolver un problema.

En general, los objetos (por ejemplo, las funciones) pueden ser denotados de tres maneras: i) de forma constructiva, indicando explícitamente sus componentes; ii) como la imagen de una metafunción (por ejemplo, la composición de dos funciones); ó iii) mediante una propiedad que satisfacen, como en *"la función que compuesta con f da la función identidad"*.

La resolución del problema 4 por Progenes requiere la combinación de sus capacidades deductiva y procedural, como veremos al final del capítulo.

Problema 13. *Hallar p y q sabiendo que $x = 1$ es un mínimo de $f(x) = x^2 + px + q$, en el que f toma el valor* 3.

Este problema utiliza la misma base de conocimiento que el anterior, junto con otra relativa a *cálculo diferencial*, cuya utilización es requerida por la presencia del concepto de *'mínimo'* en el enunciado. Por tanto, para la resolución de este problema disponemos de la definición de metafunciones como la `derivada` y el `limite` en un punto de una función real, y de metapredicados como `continua` y `derivable`, que se aplican a una función real, y `minimo`, que actúa sobre una función y un punto e indica que la función alcanza un mínimo relativo en el punto.

En este tipo de problemas en los que se buscan objetos que cumplan una cierta propiedad, PROGENES utiliza una estrategia muy común en Matemáticas, que consiste en suponer que son ciertas las condiciones, y a partir de ellas restringir los posibles valores de los objetos buscados. Si de esta manera se obtienen sólo unos cuantos valores, basta verificar cuáles de ellos verifican las condiciones impuestas inicialmente.

Esta heurística se aplica automáticamente también para resolver problemas del tipo *demostrar que* $\exists x P$, ya que una manera de demostrar la existencia de tal x es hallar un valor explícito para el mismo.

En el ejemplo, este mecanismo se utiliza en combinación con un módulo para la resolución de problemas que pueden reducirse a un sistema de ecuaciones numéricas o diferenciales. Al reescibirse las condiciones en términos de números o funciones, el módulo comprueba si esto ha dado lugar a ecuaciones que se puedan resolver, en cuyo caso se resuelven, de forma que los valores obtenidos serán los candidatos a soluciones del problema inicial. Este método se explicará con más detalle en el apartado 4.4.2 del próximo capítulo.

Problema 1. *Hallar la recta que pasa por el punto* $(0,0,3)$*, es paralela al plano* $2x - 2y - 2z + 3 = 0$*, y es tangente al paraboloide* $z = 3x^2 + 3y^2 + 2xy - 2x + 2y + 1$.

Este problema requiere una base de conocimiento correspondiente a *Geometría Euclídea*. Entre los objetos definidos en esta base de conocimiento se deben incluir `punto`, `recta`, `plano`, y `paraboloide`. Esta base de conocimiento incluye también otros conceptos como `cilindro` y `esfera`. También contiene otros más generales, como `curva`, que incluye a `recta`; `cuadrica`, que incluye a `plano`, `paraboloide`, `cilindro` y `esfera`; `superficie`, que incluye a las anteriores; y `conjunto`, que está en el extremo de esta jerarquía, y se define en la base de conocimiento utilizada en el problema anterior. Las metafunciones que se utilizan en este problema incluyen la intersección $\cap$ de dos conjuntos y el `vector_normal` a una superficie en un punto. Entre los metapredicados tenemos nuevamente $\in$, `tangente`, que opera sobre curvas y superficies, y `paralelo`, definido sobre vectores, rectas y planos.

Igual que en el primer ejemplo presentado en esta sección, para esta base de conocimiento tenemos las funciones de traducción correspondientes. Por ejemplo, una recta se representa mediante un punto y un vector, mientras que una curva se denota en general mediante una parametrización $s : \mathbb{R} \to \mathbb{R}^3$, y un conjunto se representa por un término genérico junto con una condición (un booleano). Las funciones de traducción permiten convertir por ejemplo la `recta(punto(1, 0, 0), vector(2, -1, 3))` en la `curva(funcion(`$\mathbb{R}$`,` $\mathbb{R}^3$`,` $\lambda t.(1+2t, -t, 3t)$`))`, que a su vez se puede reescribir como el conjunto $\{$`punto(`x`,` y`,` z`)` $\in \mathbb{R}^3 \mid x + 2y - 1 = 0 \wedge 3y + z = 0\}$.

Para determinar por ejemplo si el punto $(1,1,1)$ pertenece a la recta determinada por el punto $(1,0,0)$ y el vector $(2,-1,3)$, evaluaríamos la expresión:

```
punto(1, 1, 1) ∈ recta(punto(1, 0, 0), vector(2, -1, 3))
```

Puesto que la metafunción $\in$ se define para conjuntos, primero se reescribe la recta en forma de conjunto, y entonces aplicamos el procedimiento asociado a $\in$, que consiste en evaluar sobre el elemento la condición del conjunto. En este caso, resulta $1+2-1 = 0 \wedge 3+1 = 0$, que evalúa a `False`.

Este ejemplo se resuelve, igual que el anterior, suponiendo que son ciertas las condiciones que debe cumplir el objeto buscado. En este caso es necesario aplicar una heurística de expansión de objetos, que consiste en reescribir objetos compuestos como puntos, rectas o esferas, en términos de otros objetos más simples, como números y funciones reales. Este mecanismo permite reformular recursivamente determinados problemas, reduciéndolos a la realización de cálculos formales, como manipular polinomios o resolver una ecuación diferencial. En el ejemplo, la expansión del objeto buscado (una recta) permite reescribir las condiciones en forma de un sistema de ecuaciones que el sistema decide resolver, obteniendo así un valor para las incógnitas del problema, cuya validez debe ser comprobada sobre las condiciones inicialmente impuestas.

El algoritmo de expansión se describe con detalle en [SAI94], así como en [CAS94]. La solución de este problema no se mostrará en el presente trabajo, y se remite al lector a las dos publicaciones citadas para una explicación detallada de la resolución de ejemplos análogos.

3.2 Bases de conocimiento

En una base de conocimiento se distinguen dos partes bien diferenciadas. Por un lado, la base incluye las descripciones de los conceptos del dominio. Estas consisten en una definición estructural de los conceptos en términos de tipos, que determina una sintaxis para el lenguaje formal, como veremos más adelante. Como ya se ha apuntado en la sección anterior, los conceptos de la base de conocimiento se dividen en objetos, metafunciones y metapredicados. En el caso de estos dos últimos, su definición va acompañada de reglas de reescritura, que aportan una semántica procedural a ciertos

conceptos del lenguaje. En los sucesivos apartados de esta sección se verá con detalle el tipo de información que se incluye en estas definiciones.

Por otra parte, las bases de conocimiento contienen una base de reglas que expresan conocimiento en forma deductiva. Además de ser utilizadas por el motor de inferencia al modo usual, las reglas pueden ser objeto de manipulaciones por parte de las metareglas, en cuyo caso juegan el papel de conocimiento declarativo. Reglas y metareglas se codifican de forma distinta, aunque ambas son aplicadas por el mismo motor de inferencia.

3.2.1 Objetos

Los objetos describen conceptos del dominio como número `real`, `punto`, `curva`, `funcion`, `intervalo`, que responden a la idea intuitiva de *objeto*. Cada uno de ellos tiene un tipo asociado. Algunos tipos, como número `entero`, número `real`, ó `booleano` están predefinidos, y los objetos correspondientes se reconocen mediante rutinas externas (cuando aparecen como valores explícitos como 4, `5.3`, ó `True`). A partir de estos tipos básicos se definen otros más complejos, como `vector`, `superficie` ó `funcion_real`, que se construyen mediante unas componentes que a su vez pueden ser también objetos compuestos.

Por ejemplo, un cilindro se puede describir mediante dos componentes: un eje (una recta), y un radio (un número real positivo), de forma que la definición del tipo `cilindro` en la base de conocimiento tiene el siguiente aspecto:[1]

```
objeto { cilindro (eje:recta, radio:real)
         radio > 0 }
```

Con esto se declara que una expresión cuya cabecera es el símbolo `cilindro`, seguido por una expresión de tipo `recta`, y otra de tipo número `real`, define un objeto de tipo `cilindro`, siempre que la segunda componente (el radio) sea positiva.

[1]Para evitar detalles técnicos innecesarios y facilitar la comprensión de los ejemplos, la notación utilizada para las definiciones de la base de conocimiento es una versión "dulcificada", pero equivalente, del código real.

Esta declaración tiene también el efecto de definir las metafunciones `eje` y `radio`, como si vinieran dadas por:[2]

```
eje (cilindro (L, r)) := L
radio (cilindro (L, r)) := r
```

Las definiciones también se pueden utilizar en una base de conocimiento para construir incrementalmente la jerarquía de tipos, a la vez que se define un nuevo tipo. Esto se puede hacer mediante un término opcional en la definición que establece una forma de representación equivalente del objeto en la notación correspondiente al tipo superior. Por ejemplo,

```
objeto { esfera (centro:punto, radio:real)
         radio > 0
         elipsoide (centro, radio, radio, radio) }
```

define `esfera` como un caso particular de `elipsoide`, al tiempo que introduce el tipo `esfera` y las correspondientes metafunciones de acceso a las componentes de sus objetos. Asimismo, la definición da lugar a una función de traducción de `esfera` a `elipsoide` mediante la regla de reescritura:

$\texttt{TRAD}_{\texttt{esfera,elipsoide}}$`(esfera (C, r)) := elipsoide (C, r, r, r)`

al tiempo que establece la relación `esfera` $\prec$ `elipsoide` (donde $\prec$ denota la relación de orden *'subtipo de'* que determina la jerarquía de tipos).

Estas funciones se utilizan en el contexto del lenguaje formal, como explicaremos en la sección 3.3, y son esenciales para la construcción del sistema de evaluación orientado a objetos. Al mismo tiempo, permiten una utilización eficiente de la igualdad lógica entre una esfera y su correspondiente denotación como elipsoide.

3.2.2 Metafunciones

Las metafunciones son funciones como `limite`, `imagen`, `derivada`, $\cap$, `norma`, que toman como argumento objetos de un determinado tipo, y devuelven otro objeto,

[2]El signo ':=' se utiliza para expresar reglas de reescritura.

también de un tipo determinado. A las metafunciones se asocia pues una combinación de tipos de la forma $\tau_1 \times \cdots \times \tau_n \rightarrow \tau$, donde τ_j son los tipos de los n argumentos de la metafunción, y τ es el tipo del valor que devuelve. Además de una definición estructural de tipo declarativa similar a la de los objetos, las metafunciones tienen una definición procedural, es decir, se les asocia un programa que permite *calcular* (o simplificar) el valor resultante para unos argumentos dados. El código de este programa puede incluir llamadas a otras metafunciones, ya que, como se explicará más adelante, el lenguaje formal PROGENES es interpretable esto es, las expresiones PROGENES se pueden *evaluar*.

Por ejemplo, la metafunción $\cap$ se aplica sobre conjuntos dados en forma constructiva, siempre que se construyan partiendo de un mismo universo. Esto significa que un objeto representado por:

$$\{\texttt{punto}(x,\ y,\ z) \in \mathbb{R}^3 \mid x+y+z=1\} \cap$$
$$\{p \in \mathbb{R}^3 \mid \texttt{distancia}\ (p,\ \texttt{punto}(0,\ 0,\ 0)) = 1\}$$

será substitido allí donde aparezca por:

$$\{\texttt{punto}(x,\ y,\ z) \in \mathbb{R}^3 \mid x+y+z=1 \wedge \sqrt{x^2+y^2+z^2}=1\}$$

donde se ha utilizado la definición procedural de la metafunción `distancia`, lo cual ha sido posible gracias a que p, que era el primer argumento de esta metafunción, dentro de la expresión que representaba el segundo conjunto, se convierte en $\texttt{punto}(x,\ y,\ z)$. La metafunción `distancia`, igual que otras, necesita de las coordenadas de los puntos sobre los que se aplica para poder calcular el valor resultante, y cuando no dispone de ellas, permanece sin ejecutarse.

Sin embargo, un objeto representado por $A \cap \{x \in \mathbb{Z} \mid \texttt{par}(x)\}$, donde A es un símbolo de variable que denota un conjunto, no sería modificado, al no disponer el sistema de suficiente información sobre A.

La definición de una metafunción en la base de conocimiento incluye (a) una lista de argumentos, con su tipo asociado (b) el tipo del valor que devuelve la metafunción, (c) una condición que deben cumplir los argumentos para que la expresión tenga sentido, y (d) el código procedural asociado. En un problema, la mencionada condición puede dar lugar a un requerimiento que hay que verificar, o a un dato que podemos dar

por supuesto, según el contexto en que aparezca la metafunción. Además, en algunas definiciones puede aparecer un parámetro opcional de nombre `:activar`, que otorga una mayor prioridad a la aplicación de la semántica procedural de una metafunción. En los ejemplos que se muestran aquí omitiremos este parámetro para no desviar nuestra atención del propósito de este capítulo. El mecanismo de control de la aplicación de las metafunciones será descrito en el capítulo siguiente.

Así, la metafunción `composicion` se define de la siguiente manera:[3]

```
metafuncion { composicion (f:funcion, g:funcion)
              R_g ⊂ D_f
              funcion (D_g, R_f, λx.f(g(x))) }
```

De manera análoga, la `pendiente` de una recta se define como:

```
metafuncion { pendiente (recta (a:real x + b:real y + c:real = 0)
              real
              c ≠ 0
              -a/b  }
```

PROGENES admite la posibilidad de que una metafunción tenga distintas definiciones, correspondiendo a diferentes tipos de argumentos. De esta manera, es posible definir la suma de números, vectores, matrices o funciones reales, o la intersección de rectas, superficies, o conjuntos. Esto permite la utilización técnicas de Diseño Orientado a Objetos, que se describirán en la sección 3.4.

Asimismo, se pueden dar diferentes definiciones según la forma en que vengan representados los argumentos. Por ejemplo, la metafunción `imagen` de una función tiene una definición específica para el caso en que esté dada en forma explícita, y otra para una composición de funciones:[4]

[3] Recuérdese que $\mathcal{D}_f$ denota `dominio(f)`, y $\mathcal{R}_f$ `rango(f)`.

[4] Recuérdese que si `f` denota un objeto de tipo `funcion`, la notación `f(x)` es una abreviatura de la expresión `imagen(f,x)`.

```
metafuncion { imagen (funcion (d:conjunto, r:conjunto, λt.expr), x:objeto)
              objeto
              x ∈ d
              expr[x/t] }

metafuncion { imagen (f:funcion o g:funcion, x:objeto)
              real
              x ∈ Dg
              f(g(x)) }
```

La definición de metafunciones es un aspecto crucial y delicado en la construcción de una base de conocimiento. Las metafunciones permiten la compilación de conocimiento que se aplica directamente y con preferencia a las reglas de deducción.

3.2.3 Metapredicados

Como ya se ha indicado en la sección 3.1, los metapredicados vienen a ser metafunciones que devuelven un valor de tipo `booleano`. Por tanto, a cada uno le corresponde un tipo de la forma $\tau_1 \times \cdots \times \tau_n \rightarrow$ `booleano`. Por ejemplo, el metapredicado $\subset$, aplicado a dos conjuntos dados en forma constructiva, puede devolver `True` ó `False`, pero también puede devolver una expresión simbólica que puede ser manejada por el motor de inferencia.

Los predicados se definen en la base de conocimiento de manera similar a las metafunciones, salvo que no es necesario indicar el tipo del valor de retorno. Así, por ejemplo, el metapredicado `tangente` se define de la siguiente manera:

```
predicado
   { tangente (s1:superficie, s2:superficie)
     True
     ∃ p:punto | p ∈ s1 ∧ p ∈ s2 ∧ paralelo (vector_normal (s1, p),
                                              vector_normal (s2, p)) }
```

De forma similar, el predicado $\subset$ viene definido por:

```
predicado { ⊂ (c1:conjunto, c2:conjunto)
            True
            ∀ x:objeto, x ∈ c1 ⇒ x ∈ c2 }
```

Igual que con las metafunciones, es posible aplicar un mismo metapredicado a distintos tipos de objetos, como es el caso de `paralelo`, que puede actuar sobre dos rectas, dos planos, una recta y un plano, etc. También pueden ser definidos simultáneamente para distintas formas de los argumentos. Por ejemplo, el predicado $\subset$, definido más arriba, tiene una definición según la cual, si dos conjuntos vienen dados en forma constructiva en un mismo universo, la inclusión de uno en el otro es equivalente a una implicación entre los predicados que definen los dos conjuntos.

Por último, algunas definiciones involucran a predicados y metafunciones. Por ejemplo, una de las definiciones del predicado $\in$ es:

```
predicado { ∈ (x:objeto, c1:conjunto ∩ c2:conjunto)
            True
            x ∈ c1 ∧ x ∈ c2 }
```

que define una regla de reescritura que se aplica de forma habitual en los problemas de Teoría de Conjuntos.

Estas definiciones contienen metaconocimiento que una persona que estudie la materia puede adquirir en unas horas de trabajo, y PROGENES ofrece los medios para codificarlos de forma natural. Los medios para utilizarlas se verán más adelante.

3.2.4 Funciones de control

Además de objetos, metafunciones y predicados, que corresponden a conceptos del dominio, PROGENES incluye funciones de control como `hallar` y `demostrar`, que no tienen realmente un significado matemático, sino que corresponden a conceptos del *metadominio* matemático: la resolución de problemas. Estas funciones son responsables de decidir de qué manera se ha de llevar a cabo el proceso de resolución. Mientras que objetos, metafunciones y metapredicados se construyen a partir de conocimiento del dominio del problema, las funciones de control aplican metaconocimiento a la resolución

del problema. Esto incluye el tratamiento de expresiones que contienen conectores lógicos como $\forall$, $\Rightarrow$ ó $\exists$.

`hallar` toma dos argumentos: una lista de objetos, que pueden ser o contener variables, y una condición que éstos deben cumplir, que en particular puede ser una conjunción de condiciones. Los argumentos de `demostrar` son dos predicados: el objetivo y la hipótesis, que a menudo es una conjunción de predicados.[5] Las funciones `hallar` y `demostrar` se ocupan asimismo de activar los algoritmos de resolución principales (el motor de inferencia entre otros), y de que las expresiones que describen el estado de un problema se reevalúen cada vez que se activa una regla de reescritura nueva. Su funcionamiento se ilustrará en las dos últimas secciones, cuando se muestre la resolución de los ejemplos enunciados en la sección 3.1.

3.2.5 Bases de reglas

Hemos visto que las definiciones de conceptos en la base de conocimiento tienen un aspecto *puramente* declarativo, que consiste en la descripción de la estructura de los objetos, junto con la jerarquía de tipos. El aspecto *puramente* procedural consiste en las reglas de reescritura asociadas a las metafunciones, junto con las funciones de traducción entre tipos, así como las estrategias básicas implementadas por medio de las funciones de control. Además de la descripción de los conceptos del dominio, la base de conocimiento contiene una base de reglas que proporcionan al sistema una tercera capacidad, a medio camino entre lo declarativo y lo procedural, que denominaremos *deductiva*, para distinguirla de las dos anteriores.

En la literatura relativa a la Deducción Automática se encuentran apreciaciones contradictorias respecto a la naturaleza declarativa o procedural de las reglas, dependiendo de la perspectiva desde la que se aborde la cuestión. Mientras que los lógicos las ven como fórmulas a las que se aplican los axiomas de la lógica, y por tanto tienden a considerarlas información declarativa, los investigadores más próximos a las ciencias cognitivas las consideran procedurales [AND83]. En realidad, lo que usualmente se entiende por *regla* admite una gran variedad de formas, tanto en cuanto a su aspecto, como respecto a su uso, y su naturaleza puede ser considerada como procedural o declarativa según la interpretación que se les de.

[5] Aquí un *predicado* puede ser cualquier fórmula no necesariamente atómica, ya que los conectores lógicos se definen en PROGENES como metapredicados que toman argumentos de tipo `booleano`.

Las reglas PROGENES son de dos tipos: de *producción*, y de *acción*, también conocidas en la literatura como *acciones condicionales*. En las primeras, cuando se verifican sus condiciones se añade uno o más hechos a la base de datos. La parte derecha de las segundas, por el contrario, consiste en una acción que se ejecuta cuando se cumplen las condiciones. Conceptualmente, el primer tipo se puede ver como un caso particular del segundo, en el que la acción consiste en añadir hechos. Sin embargo, en PROGENES se representan de distinta manera, aunque ambas se aplican mediante el mismo motor de inferencia. Las variables libres (que se utilizan para la unificación) de las reglas de producción pueden estar cuantificadas universalmente de forma explícita (en la sección 3.6 se explicará la utilización de la cuantificación universal explícita), no así en las reglas de acción, cuyas variables libres no se cuantifican (más que implícitamente).

Las reglas de producción corresponden a teoremas, definiciones, u otro tipo de conocimiento propio del dominio o procedente del enunciado del problema a resolver. Su codificación en el sistema es muy semejante aquélla en que se enunciaría en un libro de texto, de modo que por claridad, las presentaremos aquí utilizando una notación próxima a la usual en Matemáticas, pero directamente equivalente a la interna. Por ejemplo:

$$f \textit{ inyectiva} \wedge g \textit{ inyectiva} \Rightarrow f \circ g \textit{ inyectiva}$$

Las reglas de acción, en cambio, son metareglas, que como se indicó anteriormente, son reglas que actúan sobre el estado del problema como objeto, y que pueden influir de forma directa en la estrategia a seguir en la resolución del problema.

PROGENES permite que el usuario asigne prioridades a las metareglas, lo cual contribuye a darles un carácter más procedural. Las reglas de producción, por el contrario, no tienen prioridades asociadas.

Las condiciones de reglas y metareglas se pueden verificar por evaluación antes de buscarlas en la base de hechos. En el caso de las reglas de producción, las condiciones se testean todas por separado, y después se intenta obtener combinaciones de las substituciones resultantes de cada una, hasta formar una instanciación global consistente para la regla. Esto permite instanciar una condición con la substitución obtenida para las otras, y verificarla por evaluación. En el caso de las metareglas, por el contrario, se respeta el orden de las condiciones, de forma que si una de ellas no se ha podido verificar, no se intenta instanciarla con la substitución correspondiente a otra condición posterior.

Es difícil clasificar de forma tajante las reglas de PROGENES como conocimiento puramente procedural o puramente declarativo. En realidad combinan aspectos de ambos tipos de conocimiento. En cierto sentido, las metareglas son "más procedurales" que las reglas de producción, dado que, por un lado contienen en la parte derecha código procedural puro, y por otro, como ya se ha dicho, suelen representar estrategias de resolución, es decir, conocimiento sobre *"cómo hacer"*. En cambio las reglas de producción pueden ser objeto de diversas manipulaciones por parte de las metareglas, o de otras funciones del sistema, jugando así el papel de *dato*, de información estática y pasiva. Además, este tipo de reglas expresan conocimiento en una forma muy próxima al modo en que aparece en la descripción del dominio, como hemos visto en el ejemplo del teorema de la composición de funciones inyectivas. Las metareglas también pueden ser bastante legibles y próximas a la representación del metaconocimiento en una persona.

Por otra parte, reglas y metareglas son ejecutadas mediante un mecanismo directo, el motor de inferencia, que actúa como intérprete, luego desde un punto de vista funcional, ambas se pueden considerar como programas respecto del lenguaje definido por el mecanismo intérprete, es decir, entidades que parten de unos datos y producen unos cambios. Es precisamente en esta doble naturaleza de las reglas donde reside la potencia de una representación del conocimiento mediante reglas, ya que permiten una implementación natural, por parecida a la representación humana, y al mismo tiempo operativa del conocimiento.

3.3 El lenguaje formal PROGENES

La descripción de conceptos de la base de conocimiento se utiliza para definir un lenguaje formal, el lenguaje PROGENES, en el que el enunciado de un problema se puede representar y manejar a lo largo de su resolución. Las definiciones de la base de conocimiento configuran el lenguaje, y una serie de rutinas se encargan de generar el lenguaje a partir de esta configuración.

Generar el lenguaje significa por una parte definir funciones que permitan *construir* y *reconocer* una expresión de un tipo determinado. Por otra, significa hacer efectiva la semántica procedural de los conceptos, así como definir proceduralmente metafunciones que permiten *acceder* a la información que contienen los objetos, de forma que las expresiones adquieren sentido en un entorno de evaluación simbólica. Como ya se indicó al

principio de este capítulo, el lenguaje PROGENES es más que un mero vehículo de representación de datos, y juega un papel activo en la resolución de los problemas. En otras palabras, el lenguaje no es sólo declarativo, sino que admite también una interpretación procedural mediante un conjunto de reglas de reescritura. Esto quiere decir que una expresión como $ax = b$ puede interpretarse como una operación a comprobar (llamando a las funciones apropiadas), o como una ecuación lineal, de forma que la expresión se trata como un objeto puramente sintáctico que no se evalúa semánticamente, sino que sólo se *"habla de él"*. La evaluación de expresiones del lenguaje formal se describe en la sección 3.5.2.

En el presente trabajo se dará el mínimo necesario de detalles sobre la sintaxis del lenguaje para entender los principios fundamentales de los mecanismos de resolución. Para una descripción técnica detallada de estos aspectos, ver [SAI94].

Las expresiones PROGENES pueden ser atómicas o compuestas. Estas últimas están formadas por una cabecera, que puede ser un constructor de objetos, una metafunción, un metapredicado o una función de control, y una lista de términos que le siguen, que son a su vez expresiones PROGENES. Los constructores de objetos son los nombres de los tipos, y se utilizan para construir objetos de un tipo determinado de forma explícita de acuerdo con los slots que se especifican en la base de conocimiento para ese tipo. Por ejemplo, de acuerdo con la definición de `esfera` mostrada en el apartado 3.2.1, una esfera se construiría mediante el símbolo `esfera`, seguido por un objeto de tipo `punto`, y otro de tipo `real`. Por ejemplo: `esfera(punto(2, 1, 3), 2)`.

Las expresiones atómicas consisten en un sólo término, que puede ser un valor canónico, o bien un símbolo de variable que denota un objeto de un tipo determinado. Son *expresiones canónicas* aquéllas a las que se han aplicado todas las reglas de reescritura posibles, y no contienen funciones de control ni variables sin ligar. Por ejemplo, `esfera(punto(-5, 2, 3), `$\frac{1}{2}$`)`, `conjunto(`x`:real, par(`x`))`, ó `723.4`, son expresiones canónicas, mientras que A`:conjunto`, ó `punto(4,` x`, 1)` no lo son. Obsérvese que una expresión canónica puede contener metafunciones, como es el caso de 'log' en la expresión `funcion_real (`x`,` $\log(x^3+2)$`)`, que es canónica. Por otra parte, cuando se dice que en una expresión canónica pueden aparecer variables ligadas, quiere decir que lo estén *dentro* de la expresión. Es decir, $\forall x . f(x) = x^2$, es una expresión canónica, pero x^2 no es canónico en sí, ya que contiene una variable x cuya ligadura procede de fuera de la expresión x^2.

Partiendo de la base de conocimiento se definen dos funciones, `TIPO` y `TRAD`, que operan sobre expresiones formales. `TIPO` devuelve el tipo mínimo de una expresión, de

acuerdo con la jerarquía de tipos. He aquí algunos ejemplos que corresponden a las bases de conocimiento para los problemas introducidos en la sección 3.1:

TIPO (punto(2, 1, 0)) $\longrightarrow$ punto

TIPO (imagen (funcion_real(x, $e^x + 3x^2$), 3)) $\longrightarrow$ real

La función TIPO también utiliza la jerarquía de tipos en cuanto a que permite que el tipo de las componentes de la expresión sean un subtipo del que aparece en la base de conocimiento, como ocurre en los siguientes ejemplos:

TIPO (tangente (plano ($x + y + z + 9 = 0$)
esfera (punto(0, 0, 1), 2))) $\longrightarrow$ booleano

TIPO (punto(3, 4, 5) $\in$ elipsoide (punto(0, 0, 0), 1, 2, 1)) $\longrightarrow$ booleano

TIPO también reconoce expresiones formadas mediante metafunciones de acceso a las componentes de un objeto:

TIPO (eje (cilindro (L, r))) $\longrightarrow$ recta

La función TRAD permite escribir un objeto bajo el aspecto que corresponde a un objeto de tipo superior. Como veremos en la siguiente sección, TRAD llama a la función de traducción que corresponde al par de tipos entre los cuales se va a realizar la operación.

3.4 Diseño Orientado a Objetos

Las representaciones del conocimiento que utiliza un experto se caracterizan por una gran versatilidad, de forma que la estructura de los objetos se adecua a los requerimientos del problema a resolver. Puede ocurrir incluso que un mismo objeto se represente de diferentes maneras en el curso de la resolución, a cuyos efectos una persona utiliza procedimientos de traducción de unas a otras. Así por ejemplo, si uno tiene que dibujar una circunferencia, utilizará el centro y el radio de la misma, mientras que si se quiere determinar su intersección con una recta, la esfera se representará por medio de la ecuación que la caracteriza. De acuerdo con esto, PROGENES incluye mecanismos que proporcionan una flexibilidad similar [CAS93e], que serán descritos en esta sección.

En Matemáticas a menudo se definen nuevos conceptos como combinaciones de otros más sencillos. Por ejemplo, un punto se compone de varios números, un polígono se caracteriza mediante una serie de puntos, un elipsoide se puede representar mediante su centro, la longitud de sus semiejes, y la matriz del cambio de coordenadas que da lugar a la ecuación canónica. Esto es aún más frecuente en áreas de la física como la Mecánica o la Electrodinámica, donde además de los objetos geométricos, se definen entidades más complejas como un muelle, una partícula cargada, un condensador o un sistema de poleas. Es por ello que resulta adecuado y natural considerar una representación del conocimiento basada en estructuras de tipo *frame*, en la que se definen tipos de objetos que se componen de una serie de *slots*, que a su vez pueden ser objetos compuestos, tal y como se ha descrito en la sección 3.2.

Por otro lado, puede existir una jerarquía natural entre los objetos así definidos, como por ejemplo:

```
cuadrado ≺ rectangulo ≺ paralelogramo ≺ cuadrilatero ≺ poligono
cuadrado ≺ rombo ≺ paralelogramo
           rombo ≺ poligono_regular ≺ poligono

natural ≺ entero ≺ racional ≺ real ≺ complejo

esfera ≺ elipsoide ≺ cuadrica ≺ superficie ≺ conjunto
```

Esta jerarquía determina un orden parcial entre los tipos de objetos, en forma de grafo acíclico orientado, que no es un árbol porque se dan casos en que un tipo tiene más de un padre inmediato, como `cuadrado` ≺ `rectangulo`, y `cuadrado` ≺ `rombo`, donde `rectangulo` y `rombo` no están relacionados jerárquicamente entre sí. A su vez, tenemos `rombo` ≺ `paralelogramo`, y `rombo` ≺ `polígono_regular`. Este tipo de organización jerárquica de los conceptos se refleja en las definiciones de objetos en las bases de conocimiento de PROGENES, como hemos visto.

Otros conceptos, como derivada, integral, producto escalar, operaciones aritméticas, área, etc., no responden a la idea de objeto, sin más bien a la de función (en este caso, metafunción), puesto que *se calculan*, es decir, a determinados objetos hacen corresponder determinado valor. En consecuencia, se representan en PROGENES como metafunciones a las que se asocia el procedimiento de cálculo correspondiente.

En Matemáticas se utiliza constantemente la abstracción de datos para operaciones

de este tipo, de acuerdo con las jerarquías mencionadas más arriba. Si una operación tiene sentido sobre objetos de un tipo, también lo tiene para objetos de cualquier tipo subordinado, y a menudo se definen distintos procedimientos de cálculo para cada uno de los diferentes tipos. Por ejemplo, existen diversas fórmulas para calcular el área de distintos tipos de polígonos. De forma análoga, se define en PROGENES una metafunción area, con un procedimiento para cada figura geométrica:

```
metafuncion { area (cuadrado (a:punto, b:punto))
              real
              True
              1/2 distancia(a, b)^2 }

metafuncion { area (paralelogramo (a:punto, b:punto, c:punto)
              real
              True
              distancia(a, b) × altura (a, b, c)
```

`altura (c, a, b)` representa la longitud de la altura del segmento `[a,b]` respecto del vértice `c`. Obsérvese que dos vértices opuestos bastan para determinar un cuadrado, igual que tres consecutivos son suficientes para definir un paralelogramo. En ambos casos se podría haber elegido una representación de cuatro puntos `a`, `b`, `c` y `d`, añadiendo condiciones en la definición de estos objetos. En el caso del paralelogramo, impondríamos que los lados sean paralelos dos a dos:

$$\vec{ab} \parallel \vec{dc} \wedge \vec{ad} \parallel \vec{bc}$$

y en el cuadrado, bastaría con exigir que dos lados adyacentes sean perpendiculares y de igual longitud, ya que, al ser `cuadrado` subtipo de `paralelogramo`, heredaría las restricciones que afectan a éste (que se aplicarían compuestas con la función de traducción correspondiente).

El sistema incluye también las definiciones correspondientes para rectángulos, rombos, y otras figuras, como triángulos, círculos, o la región limitada por la gráfica de dos funciones entre dos puntos. De esta forma, las metafunciones de PROGENES pueden actuar como *métodos*.

Es posible también definir métodos sobre tipos que no guardan una relación jerárquica entre sí, como es el caso de la suma de números, de vectores, de matrices o de funciones, de tal forma que el concepto de *suma* se utiliza como una abstracción de lo que en rigor son distintas operaciones, ya que en este caso no se trata de un método definido sobre una línea jerárquica, sino de una operación que se aplica en universos distintos (los tipos citados no son subtipos unos de otros). PROGENES permite sobrecargar las metafunciones, es decir, utilizar el mismo nombre para metafunciones que actúan sobre tipos que no tienen nada que ver entre sí:

```
metafuncion { + (vector (x1:real, y1:real, z1:real),
                 vector (x2:real, y2:real, z2:real))
              vector
              True
              vector (x1 + x2, y1 + y2, z1 + z2) }

metafuncion { + (f:funcion_real, g:funcion_real)
              funcion_real
              True
              funcion_real (x, f(x) + g(x)) }
```

[6]

El sistema incluye definiciones análogas para la suma de números y matrices.

La posibilidad de que un método no tenga una definición específica para un subtipo (o combinación de subtipos, caso de haber más de un argumento) determinado hace que debamos ser capaces de reescribir un objeto en la forma que corresponde al supertipo del mismo sobre el cual se define la metafunción. De acuerdo con esta necesidad, PROGENES permite que se definan funciones de traducción entre tipos de la jerarquía. Las funciones de traducción se definen para objetos dados en forma constructiva (con todos sus slots), y hacen posible la utilización del paradigma clásico de métodos y abstracción de datos.

Para aplicar una metafunción $\mathcal{F}$ (lo mismo se haría para aplicar metapredicados) a los objetos $x_1, \ldots, x_n$, se calculan los tipos mínimos respectivos de éstos, $\tau_1, \ldots, \tau_n$. Entonces se consideran aquéllas definiciones de $\mathcal{F}$ que correspondan a tipos compatibles con éstos, es decir, las $\mathcal{F}_i$ de dominio $\tau_1^i \times \ldots \times \tau_n^i$, de forma que $\tau_j \prec \tau_j^i$ para todo

[6] Recuérdese que `f(x)` es una abreviatura para `imagen(f, x)`.

j. De entre $\{\mathcal{F}_i\}$ se selecciona la más específica,[7] que llamaremos $\mathcal{F}_0$ definida sobre $\tau_1^0 \times \cdots \times \tau_n^0$. Si $\texttt{TRAD}_{\tau_j,\tau_j^0}$, que abreviaremos como $\texttt{TRAD}_j$, es la función de traducción entre τ_j y τ_j^0, entonces[8,9]

$$\mathcal{F}(x_1,\ldots,x_n) \longrightarrow \mathcal{F}_0(\texttt{TRAD}_1(x_1),\ldots,\texttt{TRAD}_n(x_n)) \longrightarrow \cdots$$

donde los puntos suspensivos de la derecha indican el resultado de ejecutar el código asociado a $\mathcal{F}_0$. Si una metafunción se aplica a una combinación de objetos para los que hay una definición específica, no habría obviamente necesidad de traducir los objetos. Dicho de otra forma, $\texttt{TRAD}_{\tau,\tau}$ es la identidad cualquiera que sea τ.

En la práctica PROGENES utiliza una sola función genérica de traducción TRAD, que según sean los tipos sobre los que actúa equivale a una u otra función de traducción específica, es decir, $\texttt{TRAD}(x,\tau) \rightarrow \texttt{TRAD}_{\texttt{TIPO}(x),\tau}(x)$, donde la función TIPO es la rutina descrita en la sección 3.3 que devuelve el tipo mínimo de una expresión. Como ya se indicó en el apartado 3.2.1, esta función se define a partir de reglas de reescritura que se incluyen en las definiciones de los objetos en la base de conocimiento.

La metafunción $\cap$, por ejemplo, se define a nivel de conjuntos, de modo que para calcular la intersección de un plano y una esfera, el sistema haría lo siguiente:

$$\texttt{plano } (ax+by+cz+d=0) \;\cap\; \texttt{esfera (punto(0, 2, 1), 5)}$$

$$\downarrow$$

$$\{\texttt{punto}(x,\ y,\ z) \in \mathbb{R}^3 \mid ax+by+cz+d=0\} \cap$$
$$\{\texttt{punto}(x,\ y,\ z) \in \mathbb{R}^3 \mid x^2+(y-2)^2+(z-1)^2=25\}$$

$$\downarrow$$

$$\{\texttt{punto}(x,\ y,\ z) \in \mathbb{R}^3 \mid ax+by-cz+d=0 \wedge x^2+(y-2)^2+(z-1)^2=25\}$$

[7]Los conflictos que puedan ocurrir en esta selección se resuelven según el orden de aparición de las definiciones en la base de conocimiento. En los casos que hemos tratado no existe en la práctica posibilidad de conflicto, por lo que no ha sido necesario aplicar este criterio de selección.

[8]El signo "$\overset{d}{\longrightarrow}$" se utiliza para indicar sucesivos pasos de evaluación. En algunos casos se indican expresiones intermedias con intención de facilitar la explicación del proceso, aunque en la práctica no llegan a aparecer explícitamente al evaluar las expresiones.

[9]Obsérvese que aquí $\mathcal{F}(\ldots)$ es una expresión PROGENES legítima, y no se trata de una abreviatura, ya que $\mathcal{F}$ no es un objeto de tipo `funcion`, sino una metafunción, y por lo tanto un verdadero símbolo funcional del lenguaje.

Si los coeficientes a, b, c y d de la recta fueran valores numéricos explícitos, o *constantes* (ver sección 3.6) del problema, la metafunción $\cap$ resolvería el sistema:

$$\left.\begin{array}{r} ax + by - cz + d = 0 \\ x^2 + (y-2)^2 + (z-1)^2 = 25 \end{array}\right\}$$

respecto de x, y y z, resultando dos puntos, un punto, o el conjunto vacío.

Para construir un sistema de traducción completo sería suficiente definir las funciones de traducción entre tipos inmediatos, de forma que la conversión entre tipos más distantes se realice mediante la composición de las sucesivas funciones de traducción entre los tipos intermedios. No obstante, existe la posibilidad de definir traducciones directas sin pasar por los tipos intermedios.

En la práctica basta definir las funciones de traducción necesarias de acuerdo con los métodos que se definan, de forma que queden cubiertos los casos en que un subtipo carece de una definición propia. De hecho, el programador de la base de conocimiento tiene libertad para decidir en cada caso entre escribir varias definiciones para los distintos subtipos, o definir las funciones de traducción correspondientes. En ocasiones lo primero es más eficiente, y más parecido a la realidad, como es el caso del metapredicado `paralelo`, al que corresponden distintos procedimientos para rectas, planos, etc. Tampoco sería eficiente, por ejemplo, escribir un cuadrado como la superficie encerrada entre la gráfica de dos funciones, para calcular su área mediante una integral. En otros casos puede resultar cómodo escribir una sola definición general. Por ejemplo, el tipo `conjunto` tiene un sinnúmero de tipos subordinados: `plano`, `recta`, `esfera`, `superficie`, `intervalo`, etc. En lugar de dar otras tantas definiciones para las metafunciones que se definen sobre conjuntos como $\in$, $\subset$, $\cap$ ó $\cup$, resulta en general más cómodo y mucho más breve definir la traducción de los subtipos al tipo `conjunto`, en especial si tenemos en cuenta que las metafunciones con varios argumentos dan lugar a muchas posibles combinaciones.

No obstante, el hecho de que haya una definición tanto para un tipo general como para uno de sus subtipos es compatible con el hecho de que exista una función de traducción entre ambos. Como ya se ha explicado más arriba, el sistema busca primero una definición específica, y caso de no haberla, aplica las funciones de traducción. Si tampoco existiera función de traducción (caso éste que no tendría mucho sentido), el sistema dejaría indicada la llamada a la metafunción, al no existir posibilidad de calcularla.

3.5 Técnicas procedurales

El código procedural de las metafunciones está basado en definiciones matemáticas, propiedades, y procedimientos de cálculo de los conceptos que representan. Este tipo de codificación del conocimiento tiene la ventaja de ser más eficiente y fácil de aplicar, al tiempo que resulta más natural y apropiado en ciertos casos, por ser más parecido a la forma en que lo representa un experto, lo cual facilita tanto su implementación como su comprensión por una persona. Por ejemplo, no es probable (aunque sería posible) que un matemático resuelva un sistema de ecuaciones aplicando teoremas, o que se plantee la decisión de calcular o no los autovalores asociados a la ecuación de una cuádrica cuando tiene que hallar los elementos característicos de ésta a partir de aquélla. Asimismo, cuando se utiliza un argumento como *"para demostrar $P \Rightarrow Q$, suponemos que P es cierto, e intentamos demostrar Q"*, no se piensa realmente en axiomas de la lógica, o en tablas de verdad.

En general, si se tiene de antemano una idea clara de cómo obtener un resultado, las personas operan proceduralmente. La experiencia juega aquí un papel importante, ya que una parte esencial del proceso de aprendizaje consiste en la adquisición de automatismos, es decir, la *compilación* del conocimiento, inicialmente adquirido en forma declarativa [AND83]. El experto se caracteriza por tener una gran cantidad de conocimiento en forma procedural, lo cual le permite alcanzar soluciones de una manera cada vez más directa, realizando ciertas tareas prácticamente sin pensar, pudiendo de esta manera concentrar su capacidad intelectual en problemas más difíciles.

Esta es la idea en la que se basa PROGENES. En las bases de conocimiento se reflejan las habilidades de un experto, más que la capacidad de aprendizaje de un alumno, con el fin de desarrollar un sistema eficaz. Se trata de representar el modelo de alguien que ya ha adquirido el conocimiento del dominio junto con la habilidad de manejarlo, lo cual implica una importante componente de conocimiento procedural. Sin embargo, sigue siendo importante la capacidad de razonar de una manera más flexible para resolver los problemas que escapan de los métodos conocidos de antemano. PROGENES ofrece la posibilidad de utilizar otros recursos cuando el conocimiento procedural no es suficiente para tratar un problema, como veremos más adelante. Esta combinación de capacidades es uno de los aspectos más potentes del sistema, y quedará ilustrado en los ejemplos que se dan al final del capítulo.

3.5.1 Localización del conocimiento procedural

Como hemos visto en la sección 3.2, las metafunciones y los metapredicados tienen una definición procedural, es decir, un programa, que permite calcular su valor cuando los argumentos tienen una forma determinada. Por ejemplo, la metafunción `producto_escalar` aplicada a dos vectores dados en forma explícita se puede calcular:

`producto_escalar (vector (2, -1, 0), vector (1, 1, 5))` $\longrightarrow$ `1`

Sin embargo, si u y v son variables que denotan objetos de tipo `vector`, tendríamos:

`producto_escalar` (u, v) $\longrightarrow$ `producto_escalar` (u, v)

es decir, la metafunción no podría calcularse, al no disponer de las componentes de los argumentos. Si éstas aparecen, aunque sean variables, el producto se puede realizar:

`producto_escalar (vector` (x, y, z)`, vector (6, 0, 1))` $\longrightarrow 6x + z$

pero las metafunciones + y × no pueden efectuarse del todo y por lo tanto se dejan indicadas. Este tipo de evaluación simbólica es semejante al de sistemas de manipulación algebraica como Mathematica [WOL91].

En el próximo capítulo veremos que algunas metafunciones se aplican inmediatamente en cuanto aparecen, mientras que otras sólo lo hacen cuando no queda ninguna regla que aplicar. Esto permite distinguir entre conocimiento procedural *de aplicación directa*, y *de aplicación controlada.*

Las funciones de control como `hallar` y `demostrar` son totalmente procedurales, aunque también forman parte del lenguaje formal, igual que las metafunciones. Estas funciones aplican estrategias muy generales a la resolución de los problemas. `demostrar` aplica al objetivo (su primer argumento) una serie de reglas de preprocesamiento del tipo *split* de Bledsoe [BLE71], y a continuación lo sitúa en el item **OBJETIVO** de la base de datos. La hipótesis (segundo argumento) es también añadida a la base de datos por medio de la función del sistema `ASERTAR`, que forma parte del motor de inferencia, y se encarga de preprocesar y clasificar las fórmulas antes de añadirlas al item apropiado.

Como vimos en el apartado 3.2.4, la función de control `hallar` toma como argumento un objeto buscado y unas condiciones que éste debe cumplir. `hallar` extrae

las variables no conocidas del objeto a buscar y las añade al item **INCOGNITAS** de la base de datos, junto con el propio objeto buscado. Las condiciones se sitúan en el objetivo, y además se añaden a un item especial **CONDICIONES**, que se utiliza para hacer razonamiento basado en el objetivo, mecanismo que se explicará brevemente en el apartado 3.6.5, junto con el papel del mencionado item.

Por último, el motor de inferencia es la componente puramente procedural inevitable en todo sistema basado en reglas. En PROGENES se ha extendido el algoritmo usual de inferencia para incluir el tratamiento de los conectores lógicos de las fórmulas que se añaden a la base de datos (función ASERTAR), así como la manipulación de hechos y reglas con cuantificación explícita. Estos mecanismos se explicarán con más detalle en la sección 3.6. Asimismo, el motor de inferencia incluye un mecanismo para el tratamiento de la igualdad mediante clases de equivalencia, de tal manera que las propiedades simétrica y transitiva se aplican implícitamente de forma sistemática, y de manera computacionalmente eficiente.

El algoritmo de unificación incluye un chequeo de la consistencia de tipos, de forma que una variable libre puede unificar con otra expresión sólo si sus tipos respectivos τ_1 y τ_2 son compatibles, esto es, $\tau_2 \preceq \tau_1$.[10] PROGENES utiliza también una forma limitada de unificación algebraica, así como un resolvedor funcional, que permite que una expresión como $f(s_1)$, donde f es una variable que denota un objeto de tipo `funcion_real` y s_1 es una constante de tipo `real`, unifique con otra expresión cualquiera *expr* de tipo `real` que contenga una variable libre t real, mediante la substitución $\{[s_1/t], [\lambda t.expr/f]\}$. Esto quiere decir que una expresión como $2y^3$ puede ser vista como una función en y, de forma que por ejemplo, el hecho $\forall y \in \mathbb{R}, f_1(y) < 2y^3$ unifica con el patrón $f(x_1) < g(x_1)$, donde f y g son variables libres y x_1 es una constante, mediante la substitución $[x_1/y, f_1/f, \lambda y.2y^3/g]$.

Como veremos más adelante, la unificación trata de manera especial las fórmulas con cuantificación explícita, tomando como variables unificables aquéllas que están bajo la cuantificación. La cuantificación puede ir acompañada de una condición de pertenencia a un conjunto, en cuyo caso el propio unificador se encarga de comprobar que la variable se instancie sólo con objetos que también pertenezcan al conjunto, información que intenta comprobar por evaluación, y buscando en la base de hechos si es necesario (pero no lanzando una demostración).

Así pues, vemos que parte de los mecanismos de resolución toman la forma de

[10] $\tau_2 \preceq \tau_1$ significa $\tau_2 \prec \tau_1$ ó $\tau_2 = \tau_1$.

rutinas que se implantan en el propio motor de inferencia, lo cual agiliza el proceso de resolución, obviando automatismos como los que acabamos de describir.

3.5.2 Aplicación del conocimiento procedural: evaluación

PROGENES aplica el conocimiento procedural a través de un sistema de evaluación y reglas de reescritura. A este respecto, el lenguaje formal es *evaluable.*. Esto quiere decir que las expresiones PROGENES se pueden interpretar de manera procedural, de tal forma que la expresión original se substituye por el valor resultante. De esta manera se puede invocar el código asociado a las metafunciones en el curso de la resolución de un problema.

El algoritmo de evaluación PROGENES es recursivo. Esto quiere decir que para evaluar una expresión, primero se evalúan los términos que siguen a la cabecera, y después se aplica la definición procedural de ésta, si la hubiera. Esto permite ejecutar las metafunciones por muy anidadas que aparezcan dentro de una expresión. Los constructores de objetos carecen de definición procedural y en consecuencia no producen ningún efecto sobre sus componentes, de forma que la expresión no varía, salvo por la substitución de las componentes por su valor.

Por ejemplo,

punto(2, 3, 1) $\longrightarrow$ punto(2, 3, 1)

esfera (punto($\sqrt{3}$, 0, 0),
distancia (punto($\sqrt{3}$, 0, 0),
plano($x + y + z + \sqrt{3}d = 0$)))
$\longrightarrow$
esfera (punto($\sqrt{3}$, 0, 0), $1 + d$)

El paradigma orientado a objetos descrito en la sección 3.4 forma parte del sistema de evaluación, y permite aplicar distintas definiciones de una misma metafunción según sean los tipos de los argumentos que recibe, haciendo una traducción de éstos a tipos superiores si fuese necesario.

3.6 Técnicas deductivas

Como ya se ha indicado anteriormente, PROGENES es capaz de combinar razonamiento automático basado en reglas y rutinas de cálculo formal, y hemos explicado también que si se conoce una forma de obtener un resultado directamente, el sistema, igual que haría una persona, opera de forma procedural. Sin embargo en ocasiones estos métodos no funcionan, y tenemos que utilizar un tipo de razonamiento menos *enfocado*, más general pero también más ineficiente, que es el razonamiento basado en reglas, intentando distintas deducciones, tomando decisiones, investigando distintas posibilidades, tratando de descartar tantas como sea posible utilizando estrategias basadas en el *know-how* de un experto sobre cómo se debe abordar el problema. Si en la sección 3.5 decíamos que PROGENES recurre a la deducción cuando las técnicas procedurales no bastan para resolver un problema, PROGENES se puede ver también como un sistema basado en técnicas deductivas que tiene además la posibilidad de aplicar procedimientos a los problemas, o a partes de éstos, cuando es posible.

PROGENES tiene muchas de las características de un sistema experto. Desde un punto de vista conceptual, está basado en el conocimiento y la experiencia que una persona adquiere en una cierta disciplina al cabo de unos años de práctica. Parte de este conocimiento se codifica en la base de conocimiento. Funcionalmente, PROGENES lleva a cabo operaciones deductivas por medio de un motor de inferencia que actúa sobre una base de datos que contiene una representación del estado del problema durante el proceso deductivo, junto con conocimiento general del dominio.

3.6.1 Base de datos

El conocimiento relevante para el problema se activa mediante un mecanismo de selección, dando lugar a definiciones y procedimientos asociados a los conceptos involucrados en el problema, y creando estructuras que se incluyen en la base de datos, como reglas, metareglas y hechos universales. La base de datos incluye los siguientes items:

- **HECHOS:** una base de hechos, que contiene las hipótesis del problema que no son reglas. Este item se divide a su vez en dos subitems: hechos *universales* y hechos sin cuantificación. Los primeros son hechos con variables cuantificadas universalmente, y se distinguen de los demás hechos en que pueden ser instanciados por objetos del problema. También se pueden utilizar como hechos ordinarios,

a cuyos efectos el algoritmo de unificación toma nota de las variables universales, e ignora el signo $\forall$.

- **REGLAS:** una base de reglas, que contiene las reglas de la base de conocimiento que han sido activadas. Como ya ha sido comentado en la sección 3.2.5, las reglas de producción pueden estar cuantificadas universalmente de forma explícita, lo cual permite que igual que los hechos universales, puedan ser instanciadas (sus variables) por constantes del problema. La instanciación de reglas por objetos no se hace exactamente igual que la de los hechos universales, como veremos en el apartado 3.6.8. Las metareglas carecen de cuantificación explícita, por lo que sólo se aplican por *modus ponens*, y no se pueden utilizar en backward, al ser su parte derecha una acción y no un hecho.

- **OBJETIVO:** un hecho a demostrar.

- **CONSTANTES:** una lista de objetos que son o se consideran conocidos. Pueden ser variables, acompañadas por sus tipos, u objetos atómicos, como números reales concretos.

- **INCOGNITAS:** una lista de objetos buscados del problema. Pueden ser variables procedentes de la eliminación de un cuantificador existencial en el objetivo, como veremos más adelante (en cuyo caso se indica su tipo), o expresiones compuestas o atómicas cuyo valor se requiere expresamente en el problema, ya sea directamente en el mismo enunciado, o como consecuencia de una decisión del sistema. Las variables libres que aparezcan en los objetos buscados también se introducen en este item. Si, como consecuencia de la aplicación de alguna regla de reescritura, una incógnita se substituyese por una expresión que no contiene incógnitas, ésta se elimina de la lista de incógnitas, y la solución se añade al item **SOLUCIONES**.

- **SOLUCIONES:** los valores hallados para los objetos buscados. Estos valores se obtienen fundamentalmente como consecuencia indirecta de la instanciación del objetivo por hechos o, de forma más directa, por resolución de ecuaciones, lo cual incluye el caso trivial en que la incógnita aparece despejada en una igualdad que proporciona un valor válido (expresión sin incógnitas) para la misma. Cuando se resuelven ecuaciones, las soluciones, como veremos en el próximo capítulo, dan lugar a reglas de reescritura, por un lado, y por otro se añaden expresamente en este item (si el valor obtenido es válido).

- **CONDICIONES:** una lista de hechos que deben cumplir los objetos buscados. Este item se utiliza para hacer razonamiento basado en el objetivo.

El sistema utiliza asimismo un item auxiliar, **CLASE OBJETIVO**, que contiene objetos estrechamente relacionados con el objetivo o los hechos buscados. La función de este item se explicará en el capítulo siguiente.

3.6.2 Tratamiento de las fórmulas

En nuestro formalismo de inferencia, las fórmulas son preprocesadas de manera similar a las heurísticas de *split* de Bledsoe [BLE71]. Esto quiere decir que a) las conjunciones en el objetivo se descomponen en tantos subproblemas como términos tenga la conjunción, cada uno de los cuales incluye como hechos los subobjetivos anteriores; b) las implicaciones en el objetivo se eliminan, suponiendo como hipótesis la parte izquierda (es decir, añadiéndola a los hechos), y tomando la parte derecha como nuevo objetivo; c) la cuantificación universal en el objetivo también se elimina, creando un símbolo de variable del tipo especificado en la cuantificación (las cuantificaciones $\forall$ y $\exists$ siempre toman una asignación de tipo como primer argumento), al que se da el estatus de *constante*, esto es, de objeto conocido (añadiéndolo al item correspondiente), y estableciendo como objetivo el cuerpo de la fórmula; d) la cuantificación existencial en el objetivo se elimina, incluyendo las variables que liga en la lista de incógnitas del problema, y añadiendo el cuerpo de la fórmula al item **CONDICIONES** (además de, por supuesto, al objetivo); y e) la cuantificación existencial se elimina de los hechos mediante la creación de las correspondientes constantes. Como ya se indicó anteriormente, el tratamiento del objetivo lo llevan a cabo las funciones de control `prove` y `hallar`, mientras que el de los hechos lo realiza una función del motor de inferencia. Estas heurísticas de reescritura no sólo se aplican al principio de la resolución, sino también cada vez que aparece un hecho nuevo.

Así por ejemplo, para demostrar $\forall x, P(x) \Rightarrow Q(x) \wedge R(x)$, se crea una constante x_1, se asume la hipótesis $P(x_1)$, se demuestra $Q(x_1)$, y en caso de éxito se añade este hecho a la lista de hipótesis y se demuestra $R(x_1)$.

También se llevan a cabo algunas simplificaciones triviales, como la eliminación de `True` en disyunciones, de `False` en conjunciones, o de términos repetidos en ambas. Asimismo, $\forall x$ `True` se convierte en `True`, `True` $\Rightarrow P$ se convierte en P, etc. Sin embargo, no se lleva a cabo una normalización exhaustiva de hechos y objetivo. Por el contrario, se tiende a respetar la estructura de los mismos en gran medida, ya que con frecuencia, la forma en que vienen escritas encierra información sobre su forma de uso más conveniente. Así, por ejemplo, un hecho que consista en una implicación

se convierte en regla (aunque las reglas pueden hacer el papel de hechos, cuando son manipuladas por metareglas). Asimismo, la cuantificación universal en los hechos permanece explícita, para permitir su instanciación por objetos del problema.

Las cuantificaciones $\forall$ y $\exists$ pueden ir acompañadas de una condición de pertenencia a un conjunto en lugar de una asignación de tipo, es decir, el lenguaje PROGENES admite que se construyan expresiones como $\forall x \in A\ P$, ó $\exists x \in A\ P$. Esto debe tenerse en cuenta a la hora de eliminar los cuantificadores. Al eliminar $\forall x \in A$ en el objetivo creando la constante x_1, se añade el hecho $x_1 \in A$. Si el objetivo es de la forma $\exists x \in A\ P$, al eliminar el cuantificador el objetivo pasa a ser $P \wedge x \in A$. El motivo de situar $x \in A$ en segundo lugar se debe a que normalmente P es un objetivo más preciso que $x \in A$ y por lo tanto es más fácil obtener un valor para x demostrando P. $x \in A$ suele ser más una comprobación que un verdadero problema. Por último, al eliminar un $\exists x \in A$ en un hecho, $x_1 \in A$ se añade a las hipótesis, donde x_1 es la constante creada a partir de x. La cuantificación universal en los hechos, como ya se ha dicho, no se elimina.

Una vez efectuado el tratamiento de las fórmulas del problema, PROGENES añade a la lista de constantes los objetos canónicos atómicos que aparezcan en el problema (normalmente, números reales concretos), de forma que los hechos universales se puedan instanciar con estos objetos.

3.6.3 Condiciones de definición de los conceptos

Como vimos en la sección 3.2, la definición los conceptos, tanto de objetos como de metafunciones y predicados, incluye unas condiciones que deben cumplirse para que una expresión formada por dicho concepto tenga sentido. Estos prerrequisitos deben comprobarse en unos casos, y en otros pueden darse por supuesto.

Las condiciones de los conceptos que aparecen en el problema se tienen en cuenta después de preprocesar las fórmulas. En el caso de las metafunciones y predicados, si el concepto aparece en una subexpresión del objetivo a demostrar, las condiciones deben demostrarse, y cuando aparece en los hechos, se supone que son ciertas. En cuanto a los objetos, las condiciones se convierten en hechos, salvo que el objeto sea una incógnita procedente de la eliminación de un cuantificador existencial en el objetivo a demostrar, en cuyo caso deben ser demostradas.

Este punto se omitirá no obstante en la explicación de aquellos ejemplos que se presentan a lo largo de esta exposición en cuya demostración no es un aspecto esencial. Veremos no obstante que en algunos casos es importante tener en cuenta estas condiciones.

3.6.4 Forward Chaining

El mecanismo básico de inferencia de PROGENES es el razonamiento basado en reglas en forward chaining. El motor de inferencia aplica las reglas según el axioma de *modus ponens*, habitualmente utilizado en los sistemas expertos basados en reglas. Sin embargo, la forma en que PROGENES aplica en la práctica este principio tiene características propias.

Para verificar la aplicabilidad de una regla se comprueba en primer lugar cuáles de sus condiciones figuran en la lista de hechos (es decir, unifican con alguna hipótesis de la base de datos). Las restantes condiciones se evalúan una vez instanciadas con las substituciones resultantes de las unificaciones anteriores. El motor considera que la regla es aplicable cuando el resultado de estas evaluaciones es `True`.

Las reglas se pueden aplicar también en backward chaining cuando se agotan las posibilidades de hacerlo en forward.

3.6.5 Razonamiento basado en el objetivo

El razonamiento basado en el objetivo es la estrategia de resolución por excelencia de los problemas que piden hallar un(os) objeto(s) que cumple(n) determinadas condiciones, y es el primero en aplicarse al resolver este tipo de problemas. Básicamente consiste en suponer que se cumplen las condiciones que debe cumplir el objeto buscado, y obtener de esa manera un valor para el mismo, comprobando después que efectivamente verifique las condiciones impuestas. Una descripción de esta estrategia, bajo un punto de vista totalmente distinto, se puede encontrar en [POL65]. La heurística se ilustrará en la resolución del problema 13, pero podemos ver aquí la idea con un ejemplo sencillo. Dado el problema:

$$\texttt{hallar } (x \in \mathbb{R},\ 2x - 3 = 1)$$

la condición $2x-3=1$ se sitúa, además de en el objetivo, en el item **CONDICIONES** de la base de datos. Los elementos de este item juegan en la resolución de los problemas el mismo papel que la base de hechos, salvo que no se busca el objetivo entre ellos, y con la particularidad de que todo hecho obtenido mediante una inferencia en la que haya intervenido un elemento de las **CONDICIONES** será añadido a este mismo item, y no a la lista de hechos. De esta manera se refleja la idea de suponer que las condiciones son ciertas: podemos razonar sobre ellas y los demás hechos, obteniendo nuevas condiciones (teniendo cuidado, eso sí, de separarlas de los hechos de verdad). El razonamiento sobre las condiciones no tiene lugar hasta que se han hecho todas las deducciones posibles utilizando hechos.

En el ejemplo, a partir de la condición dada deducimos, tras despejar x mediante la rutina apropiada, que necesariamente $x=2$. Al no haber otra solución posible, substituimos este valor en el problema, de forma que el objeto buscado es 2, y la condición que debe cumplir es $1=1$, es decir, `True`, que se cumple siempre, de forma que el problema está resuelto, al ser el objeto buscado un valor canónico. Este mecanismo se ilustra en la resolución del problema 13 en la la sección 3.9.

Esta heurística se utiliza igualmente en problemas de la forma $\exists x\ P$, también con preferencia a los demás métodos. Esto no quiere decir que la heurística sea en general más útil que otras, sino que se debe a la necesidad de anticiparse a la descomposición del objetivo, que se describirá en el siguiente apartado. Por el contrario, esta heurística suele ser menos fructífera en estos problemas que el razonamiento usual, pero tampoco interfiere con éste, sino que funciona *en paralelo*: al iniciar la resolución, el objetivo P se sitúa inmediatamente en el item **CONDICIONES**, y la resolución sigue su curso normal, con el añadido de las inferencias que se realizan paralelamente sobre los elementos de este item. En esta tesis no mostraremos con detalle la utilización del mecanismo en estos casos. Para una explicación detallada del razonamiento basado en el objetivo ver [CAS93d, SAI94].

3.6.6 Backward Chaining

Las reglas se aplican inicialmente en dirección *forward*, hasta que ya no es posible aplicar ninguna, momento en que el sistema empieza a funcionar en *backward*. El motor de inferencia utiliza un mecanismo backward particular, que se diferencia del usual en que una vez se consiguen verificar las condiciones de una regla, ésta se dispara, y se inicia de nuevo el razonamiento forward. En este sentido, se puede entender que el

razonamiento backward se utiliza como medio para desbloquear la inferencia forward. Obviamente sólo las reglas de producción pueden ser utilizadas en backward, ya que la parte derecha de las reglas de acción consiste en código procedural.

Cabe preguntarse qué aporta el razonamiento backward que no se pueda conseguir con forward, es decir, cómo es posible que se obtengan por backward las condiciones de una regla cuando se han agotado las posibilidades de inferencia forward sin obtenerlas. La respuesta es que el sistema incluye una serie de mecanismos que actúan en función del objetivo. Algunos los hemos comentado ya, como el razonamiento basado en el objetivo, y otros, como las heurísticas de introducción de operadores en función del objetivo, los veremos en el capítulo 4. El propio backward chaining es una técnica más que se puede aplicar al nuevo objetivo, aportando nuevas posibilidades de obtenerlo.

Asimismo, el backward chaining proporciona una instanciación de las condiciones que puede permitir la verificación de las mismas por evaluación, cuando no se han podido instanciar con hechos. Si por evaluación no se obtiene directamente True, también se pueden utilizar todas las técnicas del sistema disponibles para demostrarlo. Por ejemplo, si la condición de una regla es *f inyectiva*, por evaluación se convierte en $\forall x, y \in \mathcal{D}_f,\ f(x) = f(y) \Rightarrow x = y$; la implicación se eliminaría, etc.

En resumen, el objetivo del problema influye sobre el curso de la demostración, de forma que situando determinado hecho en el punto de mira del sistema, se puede emplear todo el esfuerzo necesario para obtenerlo.

El backward chaining está implementado en el mismo motor de inferencia, y consiste en buscar producciones cuya parte derecha unifique con el objetivo, estableciendo como subobjetivo la conjunción de las condiciones de las reglas así seleccionadas, y así sucesivamente, de forma que la demostración de los subobjetivos permite disparar la regla.

Los subproblemas heredan la base de de datos del estado de la resolución en el que se originan (salvo el objetivo). En caso de éxito, se ejecuta la regla seleccionada con la instanciación resultante de demostrar sus premisas, es decir, se añade como hecho su parte derecha, instanciada con la substitución resultante de la verificación de la parte izquierda. Si dicha resolución fracasa, es decir, si no se consigue demostrar la parte izquierda de la regla, ésta no se aplica.

El backward chaining es un proceso costoso, ya que supone crear un nuevo estado de resolución por cada regla que se utiliza. Es por ello (entre otras razones) que no se

hace backward chaining hasta que se hayan agotado las posibilidades de hacer forward.

3.6.7 Bases de datos locales

Durante la resolución de un problema, el sistema puede definir un subproblema que se trata independientemente del problema del cual se desprende. Es lo que ocurre cuando se tratan las conjunciones en el objetivo, o cuando se hace backward chaining. El sistema permite también que se cree explícitamente un subproblema en una metaregla, como veremos en el próximo capítulo.

Cuando se crea un nuevo problema, es necesario diferenciar lo que es cierto en general de lo que sólo lo es en el contexto local del subproblema. Por ejemplo, para demostrar un hecho de la forma $(P \Rightarrow Q) \wedge R$, primero abordamos el objetivo $(P \Rightarrow Q)$, para lo cual asumimos P y demostramos Q. Una vez resuelto este objetivo, la hipótesis P debe ser eliminada a la hora de demostrar el otro subobjetivo R. P sólo es una hipótesis válida en el contexto en el que ha surgido, y lo mismo ocurre con cualquier información que se haya derivado a partir de P. En cambio, el hecho $P \Rightarrow Q$ sí se puede añadir (una vez demostrado) a la base de hechos que se va a utilizar para demostrar Q. Otro tanto se puede decir de los objetos e incógnitas que pueden resultar de la eliminación de cuantificadores.

En general, cuando se plantea un subproblema, se crea una base de datos local (con los mismos items que la global), donde se almacenan los hechos asumidos por eliminación de una implicación, los objetos resultantes de la eliminación de una cuantificación universal, así como todos los elementos creados por medio de inferencias en las que interviene algún elemento de la base local. Cada vez que se genera recursivamente un subproblema, se añade una nueva subbase de datos, de forma que éstas se van acumulando en una pila. El sistema razona sobre todas las bases de datos, pero a la hora de añadir nuevos elementos a alguna de ellas, tiene en cuenta a cuál de ellas pertenecen los elementos en base a los cuales infiere otros nuevos, de forma que éstos se incluyen en la subbase más local de las utilizadas en la inferencia. Una vez resuelto un subproblema, se destruye la subbase que le corresponde. En la práctica, es raro que lleguen a crearse más de dos o tres subbases.

Esto afecta también a las reglas de reescritura que se habilitan en un determinado subproblema. En el próximo capítulo se explicarán los mecanismos de activación y control de estas reglas, pero podemos adelantar que cuando la creación de una deter-

minada regla de reescritura obedece a la presencia de determinados hechos en una base de datos local, la regla de reescritura se inhibe una vez se abandona el subproblema correspondiente.

Por último, el valor obtenido en un subproblema para una incógnita se transmite a los demás subproblemas. Por ejemplo, para demostrar $\exists x\ (P \wedge Q)$, se crea la incógnita x y se empieza por demostrar P. Si de la demostración de este subproblema resulta un valor para x, éste se utiliza en la demostración de Q. Todo esto quedará ilustrado en los ejemplos que se darán a lo largo de esta tesis.

3.6.8 Instanciación universal

El mecanismo de inferencia de PROGENES permite la instanciación de hechos universales con objetos del problema contenidos en el item **CONSTANTES** de la base de datos. Los hechos universales pueden también ser instanciados por hechos al modo usual. A este respecto, el procedimiento de unificación ignora el signo $\forall$ e interpreta como libres las variables cuantificadas. Del mismo modo, los hechos universales cuyo conector principal es una implicación pueden utilizarse como reglas.

Si en la cuantificación universal la variable va acompañada de una condición de pertenencia a un conjunto, para instanciar un hecho universal o unificarlo con otro hecho, se debe comprobar que la expresión que se quiere equiparar a la variable pertenece también al conjunto. Es decir, para que $\forall x \in A\ P(x)$ se instancie con *expr*, o se unifique con un hecho como $P(expr)$, el algoritmo de unificación comprueba $expr \in A$, primero por evaluación, y después buscándolo en los hechos (pero sin lanzar toda una demostración). Sólo si se verifica este hecho se obtiene una unificación válida.

Un hecho como $\forall x \in A\ P$ es equivalente a $\forall x\ x \in A \Rightarrow P$. Sin embargo, a menudo es preferible utilizar la primera representación, ya que de esa forma a) el hecho sólo se instancia con objetos atómicos (las constantes del problema), pudiendo evitar así recursiones expansivas (cuando P es de la forma $f(x) \in B$, por ejemplo), y b) el sistema puede razonar sobre P como un hecho, y no como la conclusión de una regla, lo cual tiene ciertas ventajas cuando, por ejemplo, P es una igualdad, a la que se pueden aplicar diversas heurísticas que se explicarán en el próximo capítulo. En contrapartida, el sistema no permite razonar en backward sobre este tipo de hechos. Sí lo permite en la medida en que el objetivo de un problema se puede satisfacer por unificación con un hecho universal, y en tal caso, se debe comprobar que $x \in A$ se cumple con la

substitución resultante de la variable universal, lo cual en cierto modo es plantear un subobjetivo, haciendo una forma de backward chaining. No obstante, este backward chaining es deliberadamente limitado en cuanto a que, como se ha indicado más arriba, la condición $x \in A$ se puede buscar entre las hipótesis, pero no se emplean todos los mecanismos de demostración para obtenerla.

El usuario tiene la libertad de codificar teoremas como reglas utilizando cuantificación explícita, lo cual indica que el teorema se puede instanciar con objetos, y no sólo con hechos. La diferencia de las reglas universales respecto de los hechos universales estriba en que no se añade la regla instanciada a la base de reglas, sino que se comprueban sus condiciones, y caso de cumplirse todas (ya sea por evaluación o por unificación[11] con algún hecho), se añade la conclusión de la regla a los hechos (en caso contrario no se hace nada). Las reglas de cuantificación explícita también pueden ser utilizadas por el motor de inferencia al modo usual en el razonamiento basado en reglas, ya sea en forward o en backward. En tal caso, el motor de inferencia lee las variables que aparecen en la cuantificación, para saber que símbolos debe considerar como variables a efectos de la unificación. En las reglas sin cuantificación explícita (en particular, las metareglas), las variables se reconocen por un prefijo específico.

3.7 Recapitulación

Hemos visto hasta aquí los mecanismos principales que PROGENES utiliza en la resolución de problemas, que básicamente son: evaluación directa, preprocesamiento de fórmulas, razonamiento forward, razonamiento basado en el objetivo, razonamiento backward y evaluación controlada. El sistema los aplica en este mismo orden de preferencia, y no recurre al mecanismo siguiente hasta que no pueda utilizar más el mecanismo anterior.

[11] En este caso igualdad, al estar todas las variables instanciadas

Si como consecuencia de la aplicación de uno de estos mecanismos, se crean posibilidades de utilizar otro más preferente que había quedado momentáneamente agotado, se vuelve a utilizar éste último. Es decir, si por ejemplo en un momento dado no quedan reglas que aplicar en forward ni en backward, se empiezan a activar metafunciones de aplicación controlada. Si, como consecuencia de la evaluación de determinadas metafunciones, se instancian reglas de nuevo, el sistema deja de activar metafunciones y reanuda el razonamiento basado en reglas. El control de la activación de conocimiento procedural se describirá con más detalle en el capítulo siguiente.

3.8 Resolución del problema 4

A continuación, a modo de ilustración de las técnicas descritas en este capítulo, veremos la resolución, por el sistema, del Problema 4, enunciado al principio de este capítulo. Otros problemas más complejos pueden ser resueltos utilizando los mismos o similares mecanismos.

En la explicación de este problema y de los que se irán dando a lo largo de este trabajo, se mostrará la línea principal del razonamiento, es decir, la que conduce a la solución, y se omitirán o se comentarán brevemente las deducciones que el sistema hace pero que no son relevantes de cara a la resolución del problema.

Problema 4. *Sea* $f(x) = x^3 + 3x^2 + 3x + 2$, *y sea* g *una función real. Demostrar que si* g *es inyectiva, también lo es* $f \circ g$.

Para facilitar la explicación utilizaremos, como hemos hecho hasta ahora, una notación simplificada. El sistema sitúa (por medio de la función `demostrar` descrita en el apartado 3.5.1) los datos del problema en los items correspondientes de la base de datos:[12]

OBJETIVO : g_1 *inyectiva* $\Rightarrow$ $f_1 \circ g_1$ *inyectiva*

HECHOS :

(1) $f_1 : \mathbb{R} \rightarrow \mathbb{R}$
(2) $g_1 : \mathbb{R} \rightarrow \mathbb{R}$
(3) $\forall x \in \mathbb{R},\ f_1(x) = x^3 + 3x^2 + 3x + 2$

CONSTANTES :

f_1 : *funcion_real*
g_1 : *funcion_real*

De acuerdo con lo indicado al principio de este capítulo respecto a la notación utilizada en esta tesis, los hechos (1) y (2) son una notación abreviada para expresar $\mathcal{D}_{f_1} = \mathbb{R}$, $\mathcal{R}_{f_1} = \mathbb{R}$, $\mathcal{D}_{g_1} = \mathbb{R}$, y $\mathcal{R}_{g_1} = \mathbb{R}$, que es como se representan internamente en el sistema.

La resolución comienza por la selección del conocimiento pertinente al problema. En función de las palabras clave que aparecen en el enunciado, se carga la correspondiente base de conocimiento, en este caso la relativa a conjuntos (que incluye funciones).

Aplicando las reglas de preprocesamiento de fórmulas descritas anteriormente, la implicación se elimina del objetivo, resultando:

OBJETIVO : $f_1 \circ g_1$ *inyectiva*

HECHOS :

...

(4) g_1 *inyectiva*

[12] Los símbolos con subíndice se utilizarán para denotar constantes, es decir, símbolos de variable que denotan un objeto con estatus de conocido, para distinguirlas de las variables libres o explícitamente ligadas (por una cuantificación lógica, o una λ-expresión, por ejemplo).

Como ya hemos comentado en la resolución del problema 6 en la introducción, uno de los abusos de lenguaje habituales en Matemáticas consiste en manejar indistintamente funciones y sus imágenes. Sin embargo, en un sistema de base simbólica como PROGENES no es lo mismo f que $f(x)$, al menos a priori. Es por ello que cuando en un problema aparece en los hechos una igualdad en términos de una las dos expresiones, PROGENES añade la expresión correspondiente que falta. En este caso, a partir del hecho (3) se crea la siguiente igualdad:

(5) $f_1 = funcion\_real(x, x^3 + 3x^2 + 3x + 2)$

Esta igualdad tiene la particularidad de que uno de sus miembros, el derecho, es una expresión totalmente canónica, es decir, en ella aparece explícitamente toda la información necesaria para describir el objeto que representa. En tal caso, en vez de añadir la igualdad a la base de hechos, el sistema crea una regla de reescritura de aplicación inmediata en todo el problema, mediante la cual f_1 se substituye por su valor canónico allí donde aparezca.

$$f_1 := funcion\_real(x, x^3 + 3x^2 + 3x + 2)$$

Parece clara la conveniencia de substituir una variable por su valor, si disponemos de éste. Esto nos ahorra emplear costosos mecanismos de substitución de igualdades para realizar una manipulación que en Matemáticas es tan natural como inmediata. En el capítulo 4 se describirán éste y otros mecanismos que aplican metaconocimiento semejante al que utilizan los matemáticos y permiten acelerar la resolución del problema, dirigiéndola rápidamente hacia la solución.

Tras aplicar la regla precedente, el objetivo se convierte en:

OBJETIVO : $funcion\_real(x, x^3 + 3x^2 + 3x + 2) \circ g_1 \;\; inyectiva$

y el hecho (3) desaparece de la base de hechos, ya que al substituir f_1 por su valor, resulta:

$$\forall x \in \mathbb{R},\; x^3 + 3x^2 + 3x + 2 = x^3 + 3x^2 + 3x + 2 \longrightarrow \forall x \in \mathbb{R},\; \texttt{True} \longrightarrow \texttt{True}$$

y True se elimina de los hechos.

En este punto, el motor de inferencia busca infructuosamente reglas aplicables al estado en que se encuentra el problema, y por tanto se pone en marcha el razonamiento backward, encontrando una regla cuya conclusión unifica con el objetivo en curso. Se trata del teorema que afirma que la composición de funciones inyectivas es inyectiva:

$$f \textit{ inyectiva } \wedge \; g \textit{ inyectiva } \Rightarrow \; f \circ g \textit{ inyectiva}$$

Utilizando esta regla, se establece el subobjetivo

$$\textit{funcion\_real}\,(x, x^3 + 3x^2 + 3x + 2) \;\; \textit{inyectiva} \;\; \wedge \;\; g_1 \textit{ inyectiva}$$

que se descompone en dos subproblemas, el segundo de los cuales se satisface de inmediato, puesto que es una de las hipótesis. El primer subobjetivo requiere un esfuerzo mayor.

OBJETIVO : $\textit{funcion\_real}\,(x, x^3 + 3x^2 + 3x + 2) \;\; \textit{inyectiva}$

Al no haber en este punto posibilidades de inferencia forward ni backward, el sistema activa las reglas de reescritura asociadas a las metafunciones y metapredicados de la base de conocimiento como `inyectiva` y `composicion`. En el capítulo siguiente se explicará el mecanismo por el cual se controla el momento en que se aplican las metafunciones. La metafunción `inyectiva` se define como sigue

```
predicado { inyectiva (f:funcion)
            True
            ∀x,y ∈ D_f, f(x) = f(y) ⇒ x = y }
```

Recordemos también las definiciones asociadas a los conceptos `imagen` y `composición`, mostradas en el apartado 3.2.2:

```
metafuncion { imagen (funcion (d:conjunto, r:conjunto, λt.expr), x:objeto)
              objeto
              x ∈ d
              expr[x/t] }
```

```
metafuncion { imagen (f:funcion o g:funcion, x:objeto)
              real
              x ∈ D_g
              f(g(x)) }
```

Estas definiciones son aplicables aquí por ser `funcion_real` un subtipo de `funcion`, y para llevarlo a cabo es necesario utilizar la función de traducción correspondiente. Aplicando las definiciones, se obtiene (la hipótesis g_1 inyectiva también se reescribe, pero esto no influye en la resolución del problema, de forma que lo omitimos aquí):

OBJETIVO : $\forall x, y \in \mathbb{R},\ x^3 + 3x^2 + 3x + 2 = y^3 + 3y^2 + 3y + 2 \Longrightarrow x = y$

donde el dominio de $funcion\_real\,(x, x^3 + 3x^2 + 3x + 2)$ se ha calculado, resultando $\mathbb{R}$.

La cuantificación se elimina en el objetivo, creando las constantes x_1, y_1 de tipo `real`, y la implicación se descompone, dando lugar a:

OBJETIVO : $x_1 = y_1$

HECHOS :

...

(6) $x_1^3 + 3x_1^2 + 3x_1 + 2 = y_1^3 + 3y_1^2 + 3y_1 + 2$

CONSTANTES :

...

$x_1 \in \mathbb{R}$ [13]

$y_1 \in \mathbb{R}$

Ahora una metaregla detecta que el objetivo es de la forma $expr_1 = expr_2$, y que hay un hecho, (6), que es una ecuación que involucra a $expr_1$ y $expr_2$. Dada esta situación, el sistema llama a una rutina de resolución de ecuaciones que resuelve (6) respecto de x_1 e y_2 (esta heurística se estudiará en el próximo capítulo), obteniendo una solución $x_1 = y_1$ (las soluciones en que aparecen términos imaginarios se descartan,

[13] Aquí $x \in \mathbb{R}$ debe leerse como x:`real`.

puesto que sabemos que $x_1, y_1 \in \mathbb{R}$), que el sistema aplica en forma de regla de reescritura, haciendo que el objetivo se reescriba a True, con lo cual la demostración queda concluida.

□

Si el teorema sobre la composición de funciones inyectivas no se hubiese incluido en la base de reglas, el sistema hubiese sido capaz de resolver el problema de todas formas. En tal caso, en lugar de hacer backward chaining, se hubiese aplicado directamente la definición de inyectividad a la composición del valor de f_1 con g_1 en el estado inicial del problema, y después de preprocesar el objetivo, tendríamos:

OBJETIVO : $x_1 = y_1$

HECHOS :

...

(6) $x_1, y_1 \in \mathcal{D}_g$

(7) $(funcion\_real\,(x, x^3 + 3x^2 + 3x + 2) \circ g_1)\,(x_1) =$
$(funcion\_real\,(x, x^3 + 3x^2 + 3x + 2) \circ g_1)\,(y_1)$

REGLAS :

...

(8) $\forall x, y \in \mathcal{D}_{g_1},\ g_1(x) = g_1(y) \Rightarrow x = y$

donde igual que antes, se crea y aplica la regla de reescritura que substituye f_1 por su valor canónico, haciendo desaparecer el hecho (3). Los puntos suspensivos en el item **REGLAS** corresponden a la (larga) lista de todas las reglas activas. Obsérvese que al utilizar la definición de función inyectiva, el hecho (4) se convierte en la implicación bajo cuantificación universal explícita (8), y como tal se coloca en la base de reglas. Según lo explicado en el apartado 3.6.8, el motor de inferencia puede instanciarla con objetos del problema, pero también, como se indicó en dicho apartado, puede utilizarla como cualquier otra regla.

La definición de imagen de una composición de funciones y la de imagen de una función explícita se aplican automáticamente, de forma que el hecho (7) se convierte en:

(7') $[g_1(x_1)]^3 + 3[g_1(x_1)]^2 + 3g_1(x_1) + 2 = [g_1(y_1)]^3 + 3[g_1(y_1)]^2 + 3g_1(y_1) + 2$

No hay reglas aplicables en forward, de forma que se hace backward chaining sobre la regla (8) (comprobando que se cumple la restricción de pertenencia a conjunto $x, y \in \mathcal{D}_{g_1}$, que viene garantizada por el hecho (6)), creando el subobjetivo $g_1(x_1) = g_1(y_1)$, al que se aplica el mismo razonamiento que antes, resolviendo (7') en términos de $g_1(x_1)$.

Esta segunda demostración no plantea ninguna dificultad para PROGENES, mientras que para una persona, la posibilidad de olvidarse de g_1 y centrarse en f_1, por medio del teorema de la composición de funciones inyectivas, es una ventaja. Además, una persona, especialmente alguien poco experimentado, se sentiría mas cómodo con una ecuación en una variable, más que en términos de una expresión más compleja, como es $g_1(x_1)$, aunque esto no debería ser realmente una dificultad. En cualquier caso, parece natural y justificado incluir en la base de conocimiento sobre funciones un teorema como el citado, que aparece en la mayoría de los libros de texto.

Este ejemplo ilustra cómo la combinación de cálculos e inferencias permite resolver de forma sencilla problemas como éste, que involucran ambos aspectos.

3.9 Resolución del problema 13

Problema 13. *Hallar p y q sabiendo que $x = 1$ es un mínimo de $f(x) = x^2 + px + q$, en el que f toma el valor 3.*

Este problema se resuelve también utilizando razonamiento basado en reglas, combinado con la ejecución de procedimientos y cálculos simbólicos. Además se utiliza el mecanismo de razonamiento basado en el objetivo mencionado en el apartado 3.5.1, ya que se trata del tipo de problema en que se buscan unos objetos que deben cumplir unas condiciones. La función `hallar` sitúa las condiciones en el item correspondiente (también las coloca en el objetivo, pero esto no influye en este problema), de forma que se pueda aplicar el razonamiento basado en el objetivo. La representación interna resultante es la siguiente:

INCOGNITAS : $p, q \in \mathbb{R}$

HECHOS :
(1) $f_1 : \mathbb{R} \to \mathbb{R}$
(2) $\forall x \in \mathbb{R},\ f_1(x) = x^2 + px + q$

CONDICIONES :

(3) $alcanza\_minimo\,(f_1, 1)$
(4) $f_1(1) = 3$

CONSTANTES : $f_1 : funcion\_real$

Igual que en el problema anterior, a partir de la igualdad (2), PROGENES crea la igualdad funcional correspondiente:

(5) $f_1 = funcion\_real\,(x, x^2 + px + q)$

Automáticamente, una metaregla decide que esta igualdad se convierta en regla de reescritura, es decir, $f_1 := funcion\_real\,(x, x^2 + px + q)$, de forma que f_1 se substituye en todo el problema por la expresión que la define. Así tenemos:

CONDICIONES :

(3') $alcanza\_minimo\,(funcion\_real\,(x, x^2 + px + q), 1)$
(4') $funcion\_real\,(x, x^2 + px + q)(1) = 3$

y el hecho (2) desaparece de la base de datos al resultar $\forall x \in \mathbb{R},\ x^2+px+q = x^2+px+q$, que evalúa a `True`.

La imagen de una función en un punto se evalúa inmediatamente cuando aquélla aparece en forma explícita, de modo que (4') resulta:

(4'') $1 + p + q = 3$

Ahora se aplica una regla que dice que cuando una función alcanza un mínimo en un punto y es derivable en el mismo, entonces la derivada de la función se anula en ese punto:

$$f \text{ derivable en } x \in \mathbb{R} \ \wedge\ f \text{ alcanza un minimo local en } x \implies f'(x) = 0$$

La segunda condición de la regla se satisface directamente por el hecho (3'), y la primera, instanciada por la substitución resultante de la segunda condición, se com-

prueba de forma procedural, observando que $funcion\_real\,(x, x^2+px+q)$ es un polinomio.[14] Como consecuencia de la aplicación de la regla, se deduce la condición:

CONDICIONES :

...

(6) $(funcion\_real\,(x, x^2 + px + q))'(1) = 0$

Por evaluación automática de las metafunciones `derivada` e `imagen` se obtiene:

(6') $2 + p = 0$

Ahora tenemos que el problema se ha reducido a hallar dos números, p y q, sabiendo que verifican las ecuaciones (4") y (6'). El sistema lo detecta y decide resolver el sistema de ecuaciones respecto de p y q, obteniendo soluciones para estos valores buscados:

SOLUCIONES : $p \rightarrow -2,\ q \rightarrow 4$

Finalmente se comprueba que los valores obtenidos por razonamiento basado en el objetivo verifican las condiciones. Esto se hace definiendo reglas de reescritura correspondientes a las soluciones, como efecto de las cuales las condiciones se reescriben todas a `True`, y la resolución queda concluida con éxito.

□

[14] Recuérdese que, como se indicó en el apartado 3.2.5, PROGENES permite que las condiciones de las reglas se instancien en cualquier orden.

Capítulo 4

El metaconocimiento en PROGENES

Una parte importante de un curso de Matemáticas consiste en las definiciones, axiomas, teoremas, etc., que constituyen el conocimiento sobre el dominio en sí. Sin embargo, frente a la descripción teórica del dominio, está la forma en que se utiliza esta información en la práctica. Por ejemplo, la teoría define el concepto de norma de un vector mediante la igualdad:

$$\| (\overrightarrow{x,y,z}) \| = \sqrt{x^2 + y^2 + z^2}$$

En principio, esta igualdad se podría utilizar para substituir indistintamente el miembro izquierdo por el derecho o viceversa, y la forma en que se enuncia la definición no sugiere a priori que la igualdad deba aplicarse en un sentido determinado. Sin embargo, en la práctica, esta definición se utiliza casi siempre en una dirección fija, reescribiendo expresiones como la de la izquierda por otras como la de la derecha.

El mero conocimiento de las definiciones, propiedades y teoremas del dominio, junto con la habilidad de aplicar mecánicamente los axiomas de la lógica, no es suficiente para resolver en la práctica la mayoría de los problemas de un libro de texto de Cálculo de primer año de universidad. PROGENES, igual que MUSCADET, incluye metaconocimiento sobre cómo, cuándo y dónde aplicar el conocimiento del dominio. Esto permite tomar decisiones acertadas en el curso del proceso de resolución de los problemas, controlando la busqueda de soluciones y obteniendo demostraciones más

rápidas por medio de mecanismos naturales (y por tanto fáciles de comprender para una persona).

La utilización de metaconocimiento permite a PROGENES trabajar sobre problemas difíciles en dominios que implican un conocimiento profundo, en los que sistemas como el de Boyer y Moore [BOY88] encuentran dificultades para guiar las demostraciones. Pero al mismo tiempo, a la capacidad heurística de demostradores como MUSCADET, PROGENES añade mecanismos de reescritura basada en principios de lógica con funciones, similares a los que se utilizan en sistemas como el antes citado, con lo cual se enriquecen a su vez las posibilidades de utilizar metaconocimiento, al no vernos forzados necesariamente a una representación declarativa del conocimiento. De esta manera se pueden desarrollar heurísticas potentes, como veremos en los ejemplos que se darán en este capítulo.

En este capítulo se describirán diversas heurísticas que utiliza PROGENES, la mayoría de las cuales se lleva a cabo mediante metareglas inspiradas en el *know-how* matemático. Las metareglas son reglas que se aplican al estado de la resolución del problema, a diferencia de las reglas ordinarias, que expresan condiciones y conclusiones del nivel objeto, como la que corresponde al teorema del valor medio:

$$f \textit{ derivable en } [a,b] \subset \mathbb{R},\ c,d \in [a,b] \implies \exists z \in [a,b] \textit{ tal que } f(d)-f(c) = f'(z)\ (d-c)$$

Las metareglas por el contrario incluyen condiciones de meta-nivel, esto es, *meta-condiciones*, que expresan hechos relativos a la base de datos en sí. La parte derecha de estas reglas es una acción que puede operar directamente sobre los distintos items que describen el estado del problema. Por ejemplo, veremos más adelante una metaregla que dice que en caso de que busquemos el valor de una variable real, y existan ecuaciones que la involucren y puedan ser resueltas, entonces la solución del sistema proporciona la solución del problema. PROGENES incluye también, como veremos, otras metareglas que permiten manipulaciones similares, como eliminar operadores molestos, formar igualdades entre funciones a partir de igualdades entre imágenes y viceversa, o convertir una igualdad en regla de reescritura.

Por otra parte, algunas heurísticas están implícitas en diferentes mecanismos del sistema. Por ejemplo, las heurísticas de preprocesamiento de fórmulas y tratamiento de conectores se llevan a cabo por el motor de inferencia y la función de control `demostrar`, como vimos en el capítulo precedente. Asimismo, el motor de inferencia utiliza un

algoritmo de unificación que aplica un mecanismo de resolución funcional, y que, como ya se ha indicado, se ocupa también de la aplicación automática de las propiedades simétrica y transitiva de la igualdad, por medio de una representación de la misma en forma de clases de equivalencia. Otras heurísticas, también de tipo *built-in*, se plasman en la representación del conocimiento mediante metafunciones y metapredicados, como veremos a continuación.

El contenido de este capítulo está distribuido de la siguiente manera. En primer lugar, se analiza la naturaleza y el papel de las metafunciones, desde el punto de vista del metaconocimiento que encierran implícitamente, y el que permite controlar su aplicación. En la sección siguiente se comentan someramente los mecanismos de activación del conocimiento del sistema. A continuación (sección 4.3) se describen con detalle una serie de heurísticas (metareglas) generales, muchas de las cuales están relacionadas con el manejo de igualdades. En la sección 4.4 se presentan otras heurísticas de utilización más específica, y finalmente, la sección 4.5 ofrece una muestra de la utilidad de las técnicas desarrolladas a través de la resolución de cinco problemas.

Notación

Las metareglas que se describen en este capítulo se presentan utilizando una notación similar a la habitual para las reglas de inferencia en el cálculo de *sequents*. Las metareglas se representarán de la siguiente manera:[1]

$$\frac{\textit{condiciones}}{\textit{lista de hipotesis} \overset{\textit{incognitas}}{\underset{\textit{clase objeto}}{\vdash}} \textit{objetivo}} \quad \textit{nuevo estado resultante} \quad \textit{acciones}$$

donde:

- Bajo la línea de la regla se refleja el estado del problema en el que se puede aplicar la metaregla.
- Sobre la misma se indica el estado del problema que resulta de la aplicación de la regla.
- Sobre el signo $\vdash$ se puede indicar la lista de objetos (términos) buscados.

[1] En la práctica su codificación se hace según la forma convencional *SI condición ENTONCES acción*.

- Bajo este mismo signo, se pueden poner los elementos del item **CLASE OBJETIVO**, que se describirá en el apartado 4.3.2.

- A la izquierda de la línea que separa los dos estados del problema y en una caja, se indican (meta)condiciones necesarias para la aplicación de la regla, si las hubiera.

- A la derecha de la línea y en una caja también, se indican las posibles acciones a ejecutar cuando se cumplen las condiciones. Estas acciones se ejecutan antes de llevar a efecto el nuevo estado, de forma que en éste pueden aparecer variables cuyo valor se asigna en las acciones.

Esta notación indica que cuando el estado de la resolución de un problema es el que se indica bajo la línea de la regla, y se cumplen las *condiciones* indicadas a la izquierda, se ejecutan las *acciones* indicadas a la derecha, y se substituye el estado actual por el *estado resultante* que figura sobre la línea. En realidad, igual que en lógica, esta notación indica que la expresión que se indica sobre la línea implica la que aparece debajo, entendiendo un estado de la demostración como una fórmula en sí. Por ello, cada estado se substituye por otro del cual se deriva, de forma que si llegamos a un estado que reconocemos como tautológico, hemos conseguido una cadena de implicaciones que conectan `True` con el estado a resolver. La aplicación de estas metareglas es irreversible, es decir, no dan lugar a la generación de un árbol de demostración, sino que se suceden unas a otras.

Existe otro tipo de metareglas que sí producen ramificaciones, y que se representan con la notación:

$$\textit{estado inicial} \quad \underset{\boxed{\textit{condiciones}}}{\rightsquigarrow} \quad \textit{nuevo problema}$$

para indicar que se genera un nuevo problema a partir del actual, de forma que si resolvemos el nuevo problema queda resuelto el original. Bajo el signo $\rightsquigarrow$ se pueden indicar restricciones adicionales a la aplicación de la regla. El problema generado se trata independientemente del anterior, y en caso de que no se pueda resolver, el control retorna al problema original (*backtracking*), de forma que se pueden intentar nuevas estrategias. Al generar un subproblema, éste hereda todos los elementos de la base de datos que no aparecen explícitamente en la regla. Los que aparecen a la izquierda de $\rightsquigarrow$ se heredan sólo si se indican también a la derecha.

Por la descripción que hemos dado de los dos tipos de reglas podría parecer que las primeras son "más peligrosas" que las segundas, ya que sus efectos son aparentemente más definitivos. Sin embargo, como veremos, estas reglas aplican razonamiento *casi* monotónico, añadiendo información, sin eliminar ni modificar la existente. Las únicas excepciones son las que definen reglas de reescritura, pero incluso éstas se aseguran de que las substituciones a que van a dar lugar son totalmente convenientes. Las reglas que producen subproblemas se utilizan para realizar modificaciones importantes en el problema, sin perder la posibilidad de volver atrás. Se trata fundamentalmente de reglas que modifican el objetivo. La generación de un subestado de resolución supone un cierto coste, y es por ello que antes de aplicar este tipo de reglas, se aplican las que sólo añaden hechos (o definen reescrituras) en un cierto estado de la demostración.

El lenguaje formal PROGENES permite expresar enunciados de lógica de predicados de primer orden con igualdad y funciones. Los predicados del lenguaje son los metapredicados PROGENES. Los términos del lenguaje son las expresiones que denotan objetos. Las funciones del lenguaje son los constructores de objetos y las metafunciones, que son los que se pueden utilizar como cabecera para formar expresiones que representan objetos. A menudo emplearemos el término *operadores*, para englobar a metafunciones y constructores de objetos. No deben confundirse las funciones del (meta)lenguaje con las funciones del lenguaje objeto, es decir, los objetos de tipo `funcion`, que en ningún caso pueden jugar el papel de símbolo funcional del lenguaje PROGENES, aunque así lo haga parecer la notación que se utiliza en este trabajo, según la cual se representa como $f(x)$ lo que en realidad es `imagen`$(f,\ x)$.

Para representar objetos (términos) se utilizarán por lo general las metavariables x, y, s, t, u. Estas variables pueden ir tipadas en la misma regla, indicándose el tipo de los mismos mediante la sintaxis *objeto* : *tipo*, que indica una restricción de la aplicación de la regla al caso en que el objeto es del tipo indicado. φ y ψ se utilizarán para denotar elementos de la base de hechos, es decir, fórmulas, y los símbolos Γ, Δ se emplearán por lo general para representar un conjunto de hipótesis $\{\psi_1, \ldots, \psi_n\}$, mientras que las metavariables Σ y Υ se utilizarán pata denotar un conjunto de términos $\{t_1, \ldots, t_m\}$. Las funciones del (meta)lenguaje (metafunciones y operadores) se representarán por lo general mediante el símbolo f, el mismo que se utiliza también con frecuencia para denotar objetos de tipo `funcion`. La ambigüedad es intencionada, y pretende acercar el lenguaje y los mecanismos del sistema a los que se utilizan habitualmente en matemáticas. El contexto, y en caso de ser necesario, una explicación expresa, determinarán claramente en cada caso a cuál de las dos nociones nos estamos refiriendo. Por último, el símbolo $\mathcal{F}$ se utiliza para denotar metafunciones, y $\mathcal{R}$ para las relaciones

o predicados binarios. Este último símbolo también se utiliza en expresiones como $\mathcal{R}_f$, que es una abreviatura para rango(f), y no tiene nada que ver con relaciones binarias.

Por último, apuntar que a la izquierda del signo $\vdash$ se incluyen, junto con los elementos del ítem **HECHOS**, los del ítem **CONDICIONES**, en caso de que los haya (es decir, cuando se hace razonamiento basado en el objetivo). Como ya se explicó en el capítulo anterior, las condiciones juegan un papel análogo a los hechos, salvo que no se da por resuelto un problema cuando el objetivo aparece entre ellas (téngase en cuenta que en particular las condiciones proceden inicialmente del objetivo), de forma que, para simplificar la notación, se dará por supuesto que los mecanismos de razonamiento actúan también sobre ellas.

Igual que en el capítulo anterior, se utilizarán muchas abreviaturas, siempre equivalentes al lenguaje formal, para acercar la notación utilizada a la habitual en Matemáticas y facilitar así la comprensión de los ejemplos. Asimismo, en los ejemplos que se muestran de demostraciones del sistema, seguiremos en general la línea que conduce a la solución, tratando de recalcar los puntos más importantes de la misma, sin explicar con todo detalle, o incluso omitiendo, algunos aspectos secundarios irrelevantes, así como las deducciones que no aportan nada a la resolución, pero que el sistema también realiza, que en general no son muchas, gracias a que las heurísticas llevan la demostración en direcciones adecuadas.

4.1 Metafunciones

La continuidad de funciones de $\mathbb{R}$ en $\mathbb{R}$ se define en Matemáticas de la siguiente manera:

Si $\mathcal{U} \subset \mathbb{R}$, $\mathcal{U}$ *abierto,* $f : \mathcal{U} \to \mathbb{R}$, *y* $a \in \mathcal{U}$,

$$f \text{ continua en } a \iff \lim_{x \to a} f(x) = f(a)$$

Los axiomas de la lógica nos permiten en principio utilizar una equivalencia lógica como la anterior tanto para reescribir un hecho de la forma f *continua* por su definición, como para realizar la substitución contraria. Sin embargo, en la práctica, la definición se utiliza casi siempre en una dirección fija, de izquierda a derecha.

Otro tanto ocurre con la definición de conceptos que se definen mediante una igualdad, como por ejemplo, la pendiente de una recta:

$$\textit{pendiente de la recta } ax + by + c = 0 \;=\; -\frac{a}{b}$$

La forma habitual de utilizar esta definición es para substituir la parte izquierda por la derecha (raro sería que un matemático substituyese la expresión de la derecha por la de la izquierda). La práctica y la experiencia en el manejo de definiciones como la anterior conduce a establecer una direccionalidad en el conocimiento que expresan.

Esta es la idea que subyace en las metafunciones y metapredicados PROGENES. Como vimos en el capítulo anterior, las metafunciones tienen un código asociado, basado en la definción matemática y/o procedimientos de cálculo popios del concepto que representan. Las metafunciones se aplican por medio de un sistema de reescritura de expresiones, mediante el cual las subexpresiones cuya cabecera es una metafunción se substituyen por el valor que resulta de ejecutar el programa asociado, cualquiera que sea el nivel de anidamiento de la subexpresión. La posibilidad de utilizar conocimiento procedural asociado a determinados conceptos potencia la utilización de metaconocimiento en el sistema.

4.1.1 Direccionalidad

Las metafunciones establecen implícitamente una igualdad entre la llamada a la metafunción y el valor resultante de la ejecución del código. Es decir, si $\mathcal{F}$ es una metafunción de aridad n, y *val* es el valor devuelto para los argumentos $x_1, \ldots, x_n$ por el programa asociado a $\mathcal{F}$, se cumple la igualdad $\mathcal{F}(x_1, \ldots, x_n) = val$. Pero las metafunciones establecen además una direccionalidad en estas igualdades, en cuanto a que el concepto que representan se puede reescribir por su definición, mediante la ejecución del código procedural, pero la substitución contraria nunca se realiza. Esta direccionalidad se establece *a priori*, en base a criterios de experiencia y *know-how*.

Los metapredicados establecen lo análogo a lo anterior respecto de las equivalencias lógicas en lugar de las igualdades. Si $\mathcal{P}$ es un metapredicado, se cumple la equivalencia $\mathcal{P}(x_1, \ldots, x_n) \Leftrightarrow val$, donde *val* es el valor retornado para los valores $x_1, \ldots, x_n$ por el código procedural asociado a $\mathcal{P}$. Aquí también, la equivalencia se aplica en una dirección preestablecida. Obsérvese que los metapredicados se pueden considerar como

metafunciones de tipo $\tau_1 \times \cdots \times \tau_n \rightarrow$ `booleano`, de forma que la equivalencia se puede interpretar como la igualdad entre booleanos, y por lo tanto, todas las propiedades de las metafunciones lo son también de los metapredicados. De ahí que a menudo utilizaremos la palabra 'metafunción' para englobar ambos conceptos. La distinción entre metafunciones y metapredicados es de tipo conceptual, y procede de la idea de lo que tradicionalmente se entiende en lógica por predicados y símbolos funcionales. Aunque ambos juegan un distinto papel en el razonamiento basado en reglas, el lenguaje PROGENES garantiza que si un enunciado se escribe mediante una expresión formal bien construida, entonces la base de datos recibirá los elementos adecuados en cada item, y el motor de inferencia encontrará expresiones coherentes, de forma que la distinción formal entre ambas categorías no es estrictamente necesaria.

El demostrador de Boyer y Moore [BOY88] utiliza un principio similar al de las metafunciones PROGENES, basado en el direccionamiento de igualdades, según el cual la elección de una direccionalidad recae en el usuario. Este puede etiquetar las igualdades y equivalencias (éstas son tratadas como un caso particular de la igualdad) de la base de conocimiento, indicando con ello que deben aplicarse como reglas de reescritura en una dirección predeterminada. Es decir, el sistema no decide por sí solo la direccionalidad, sino que la establece el usuario de antemano, igual que en el caso de nuestras metafunciones. En el apartado 4.3.1, veremos cómo PROGENES es capaz en algunos casos de decidir por sí solo que una igualdad debe convertirse en regla de reescritura.

4.1.2 Naturaleza del conocimiento contenido en las metafunciones

En muchas ocasiones la igualdad (o la equivalencia) subyacente a la definición procedural de una función procede literalmente de una definición matemática. Tal es el caso de metafunciones como `pendiente`, `norma` ó `suma` de funciones, y de metapredicados como `continua`, `ortogonal`, ó $\subset$. Su definición matemática es de la forma $\forall x, \mathcal{F}(x) = expr$ para las metafunciones y $\forall x, \mathcal{P}(x) \Leftrightarrow expr$ para los metapredicados.

En algunos casos el código de las metafunciones consiste en una serie de rutinas definidas a partir de procedimientos matemáticos de cálculo asociados al concepto, como en el caso de `derivada`, `integral`, `limite` de una funcion, o `inversa` de una matriz. Los algoritmos de cálculo se definen (o se pueden definir) formalmente como parte del conocimiento del dominio, e incluso se pueden encontrar en libros de texto de

enseñanza media o universidad. En algunas ocasiones estos métodos permiten suplir, como veremos a continuación, la carencia de una definición explícita, y en otras evitan utilizar definiciones que, aunque explícitas, resultan aparatosas.

Por ejemplo, mientras que `derivada` e `integral`, tienen una definición explícita en forma de igualdad, conceptos como `limite` de una función, `inversa`, o `valores_propios` de una matriz, son definidos en Matemáticas en forma *implícita*, es decir, mediante una propiedad que verifican, que las más de las veces es una igualdad en la que aparece $\mathcal{F}(x)$, pero no se puede despejar. Así por ejemplo, la inversa A^{-1} de una matriz A es aquella matriz que cumple $A^{-1}A = I$. Los `valores_propios` de una matriz A son aquellos números λ para los que existe algún vector u tal que $Au = \lambda u$. La propiedad que se utiliza para caracterizar el concepto no tiene por qué ser una igualdad: el límite de una función f en un punto a es el número l que verifica que $\forall \epsilon > 0, \exists \delta(\epsilon)$ *tal que* $\forall x, |x - a| < \delta \Rightarrow |f(x) - l| < \varepsilon$. Las definiciones formales de estos conceptos no proporcionan en sí un buen método para determinar su valor. Sin embargo, a partir de ellas se construyen formalmente procedimientos de cálculo que permiten la obtención efectiva de resultados. Por ejemplo, para calcular los valores propios de una matriz dada, se plantea la ecuación indicada anteriormente y se exige que el sistema resultante sea compatible, condición que da lugar a una ecuación en términos de λ que permite determinar los valores buscados. De esta manera, la igualdad original queda escondida detrás de una serie de algoritmos de resolución de ecuaciones. La definición de estos procedimientos pasa a menudo por establecer una serie de teoremas sobre los que se sustentan. El cálculo de límites, por ejemplo, se basa en teoremas como el del límite de la suma, del producto, el de l'Hôpital, etc.

Los conceptos como `derivada` o `integral`, aunque tienen una definición explícita en forma de igualdad, ésta no resulta normalmente fácil de manejar, por lo que se utilizan preferentemente métodos matemáticos de cálculo que se construyen, igual que antes, a partir de la definición formal del concepto y de teoremas que se derivan de ésta. Los procedimientos de cálculo de derivadas, por ejemplo, están implícitamente basados en teoremas (derivada de la suma, del producto, regla de la cadena, etc.) en los que no reparamos cuando los aplicamos en forma de reglas de derivación.

Todos estos algoritmos de cálculo toman la forma en nuestro sistema de programas que actúan como cajas negras y llevan a cabo automáticamente los procedimientos. En este tipo de metafunciones, también se realiza una aplicación direccionada de la igualdad. La diferencia respecto de metafunciones como `pendiente` o `norma` es que aquí la igualdad (o propiedad) original, así como los teoremas que se derivan de ella, no son visibles en el código, pero están ahí, implícitamente. Por ejemplo, al derivar

una expresión como $x^2 + \sin e^x$, utilizamos las igualdades $(f(x)+g(x))' = f'(x)+g'(x)$, $(f(g(x)))' = f'(g(x)).g'(x)$, $(x^n)' = nx^{n-1}$, $(\sin x)' = \cos x$, y $(e^x)' = e^x$, que son teoremas, y se aplican siempre en la misma dirección. Es verdaderamente raro en Matemáticas que, por ejemplo, se substituya una suma de derivadas por la derivada de una suma, a menos que haya una razón especial para ello.

La construcción de procedimientos de cálculo a partir de definiciones declarativas viene a ser una forma de *compilación* del conocimiento, que no necesita realizar nuestro sistema, sino que viene ya hecha por los matemáticos. PROGENES no realiza la compilación de este conocimiento, sino que capta un conocimiento matemático ya compilado.

4.1.3 Control

Otra cuestión fundamental relacionada con las metafunciones (y metapredicados) es la elección del momento apropiado en que deben ser aplicadas. No todas las definiciones de conceptos conducen a las mismas consecuencias. Efectuar una suma de números, por ejemplo, tiende a simplificar el problema, mientras que aplicar la definición de derivada en términos de límites puede introducir elementos de mayor complejidad, aunque en ocasiones es necesario. PROGENES permite un cierto control sobre el momento en que se aplica una determinada metafunción, ya que, una vez más, la práctica establece que determinadas definiciones deben ser de aplicación inmediata o prioritaria, mientras que otras se utilizan como último recurso.

Cuál es el mejor momento para aplicar una definición procedural depende de diversos factores, como la dificultad que conlleva el manejo de los conceptos que se introducen en ella, o la cantidad de teoremas (o su relevancia) que involucran al concepto representado por la metafunción. Por ejemplo, para demostrar un hecho de la forma *f derivable en a*, antes de intentar

$$existe \lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

siempre se intenta utilizar los teoremas sobre derivabilidad, comprobando por ejemplo si f se escribe como suma, producto, composición, etc., de funciones derivables, lo cual en principio es más sencillo que demostrar la existencia de un límite.

En cambio, una expresión como $(f \circ g)(x)$ se reescribe automáticamente como $f(g(x))$, salvo en casos excepcionales en que pueda haber algún motivo para no hacerlo.

El comportamiento general de las metafunciones de PROGENES viene determinado por el siguiente mecanismo: al cargar una base de conocimiento, PROGENES lee una a una las definiciones de conceptos; en el caso de las metafunciones, su definición procedural se guarda en una lista para su activación en el momento adecuado. Posteriormente, ya en el curso de la resolución de un problema, en el momento en que no quedan reglas que aplicar, se coge la primera metafunción de la lista y se activa el procedimiento asociado, de forma que se ejecuta en todo el problema, allí donde aparezca la metafunción. Como consecuencia de los cambios producidos, es posible que se instancie alguna regla, de modo que se pone en marcha de nuevo el motor de inferencia, hasta que no se pueda utilizar ninguna regla, y así sucesivamente, hasta que se activen todas las metafunciones, o se resuelva el problema.

El comportamiento por defecto de las metafunciones permite así evitar que se pierdan posibilidades de aplicar teoremas que involucran a un concepto antes de utilizar su definición. Así por ejemplo, para demostrar que una función es continua, primero se intentan aplicar todos los teoremas posibles, comprobando si hay un hecho que diga que es derivable (para utilizar *derivable* $\Rightarrow$ *continua*), o intentando comprobar si es la suma, la composición, etc., de funciones continuas, y si todo esto falla, entonces, y sólo entonces, se aplica la definición de continuidad. Esto se corresponde con lo que habitualmente se hace en Matemáticas, y se puede ver en la resolución del problema 4 en el capítulo anterior. En este ejemplo primero se hace backward chaining con el teorema que establece que la composición de funciones inyectivas es inyectiva, y sólo después se aplica la definición de inyectividad, que de haberse utilizado antes hubiese impedido la utilización del teorema, al desaparecer los hechos de la forma *f inyectiva*, substituidos por su definición.

Activar las metafunciones una a una, en lugar de todas a la vez, permite que como consecuencia de los cambios producidos por la evaluación de una metafunción, se instancie una regla que involucra a otra que no está activada aún. Si ésta última hubiera sido activada al mismo tiempo que la primera, la regla podría no instanciarse, al desaparecer la metafunción, substituida por el valor resultante de su código.

Las definiciones almacenadas durante la carga de la base de conocimiento se ordenan automáticamente, siguiendo el criterio de retrasar la activación de metafunciones que aparecen en el código de otras. Por ejemplo, la definición de continuidad se activa antes que la de límite, ya que este último concepto aparece en la definición de aquél.

De esta forma, no se pierde la oportunidad de hacer razonamiento conceptual sobre ninguno de los dos. En segundo término, las metafunciones se ordenan de acuerdo con el número de reglas que las involucran, de forma que se activan más tarde cuanto mayor sea este número. Los conceptos de continuidad y derivabilidad, por ejemplo, aparecen en multitud de teoremas, como consecuencia de lo cual se activan lo más tarde posible. De lo descrito se desprende que el orden de las definiciones depende sólo de la información contenida en la base de conocimiento. Es por ello que el sistema incluye un programa que determina este orden de antemano y de una vez por todas, de forma que no es necesario reordenar las metafunciones cada vez que se carga la base de conocimiento.

PROGENES permite que el usuario modifique el comportamiento por defecto de las metafunciones, mediante un parámetro opcional de nombre `:activar` en la definición de las metafunciones, que indica que la metafunción debe activarse inmediatamente al cargar la base de conocimiento. De esta forma, si al leer una base de conocimiento el sistema encuentra este parámetro, la semántica procedural se hace efectiva automáticamente, y si por el contrario no se ha indicado nada, se aplica el mecanismo por defecto. PROGENES permite así decidir de antemano al desarrollador de la base de conocimiento qué tipo de comportamiento quiere imponer a las metafunciones que define, distinguiendo entre metafunciones *de aplicación directa* y *de aplicación controlada.*

En la definición de `pendiente`, por ejemplo, se especifica que su semántica procedural es de activación inmediata:[2]

```
metafuncion { pendiente (recta (a:real x + b:real y + c:real = 0)
              real
              b ≠ 0
              -a/b
              :activar }
```

lo mismo que la defínción de rectas paralelas en el plano:

[2]Recuérdese que en el capítulo precedente se omitió este aspecto.

```
predicado { paralelo (recta (a1:real x + b1:real y + c1:real = 0,
                     recta (a2:real x + b2:real y + c2:real = 0)
            True
            a1 b2 = a2 b1
            :activar }
```

mientras que la metafunción `continua` se ajusta al mecanismo general de aplicación controlada:

```
predicado { continua (f:funcion_real, a:real)
            a ∈ D_f
            lim_{x→a} f(x) = f(a) }
```

En general, las rutinas de cálculo formal son conocimiento de aplicación inmediata, mientras que las definiciones procedurales que se corresponden de forma directa con la definición teórica de un concepto pueden ser de aplicación directa o controlada, según el caso. Por ejemplo, la definición de planos `paralelos`, o los procedimientos de cálculo de la metafunción `limite`, son de aplicación inmediata, mientras que la definición de `limite` en términos de ϵ y δ es de aplicación controlada (en el capítulo anterior vimos que una misma metafunción puede tener varias definiciones procedurales). En cualquier caso, esto es decisión del usuario, de modo que éste tiene total libertad para cambiar este criterio.

Muchas metafunciones tienen unas definiciones procedurales directas y otras de aplicación controlada. Por ejemplo, la metafunción `derivada` tiene una serie de propiedades procedurales de aplicación inmediata, como las que corresponden a los casos en los que el argumento permite utilizar alguna de las reglas clásicas de derivación de expresiones matemáticas. Y por otro lado, la definición en términos de la existencia de un determinado límite es de aplicación controlada.

Una vez activada una metafunción, el procedimiento asociado se aplica sistemáticamente (siempre) y automáticamente (inmediatamente) allí donde aparezca la expresión. Así, la metafunción `pendiente`, se aplica en cuanto aparece, mientras que la definición de `continua` en términos de límites no se aplica en principio, hasta que no pueda aplicarse ninguna regla, en cuyo caso la definición es activada.

Si la activación de una serie de metafunciones se produce durante la resolución de un subproblema, como por ejemplo cuando se descompone una conjunción en el obje-

tivo, estas metafunciones se vuelven a inhibir todas al retornar el control al problema original, ya que en éste no tienen por qué haberse acabado las reglas aplicables, y por lo tanto no se justifica la activación de las metafunciones.

4.1.4 Aspectos cognitivos

A menudo la representación procedural es más adecuada para cierto tipo de conocimiento. La inversión de matrices, o el cálculo de un determinante, por ejemplo, responden a una formulación típicamente procedural, es decir, el conocimiento que un experto utiliza en estas operaciones se expresa en forma algorítmica. En un sistema de demostración que utilice exclusívamente técnicas basadas en inferencia lógica, la resolución de (sub)problemas de este tipo es costosa y antinatural. Las metafunciones de PROGENES permiten una representación no sólo más eficiente, sino también más natural del conocimiento de este tipo.

Las metafunciones juegan un papel fundamental en la resolución de los problemas, y permiten la utilización de conocimiento compilado que, una vez activado, se aplica directamente y con preferencia sobre las reglas deductivas. Un matemático o un estudiante de Matemáticas tiene muchos de estos procedimientos activados, y este conocimiento tiene un papel esencial en la agilización del proceso deductivo. La capacidad de activar el tipo adecuado de reglas de reescritura permite tanto al ser humano como a PROGENES realizar demostraciones que de otra forma llevarían mucho más tiempo.

De hecho, el proceso de aprendizaje tiene una componente importante que consiste en crear las reglas de reescritura adecuadas a partir de las reglas de la lógica. Como consecuencia, uno puede distinguir diferentes niveles de comprensión en una persona en distintos momentos, dependiendo de las operaciones que aplica sistemáticamente. Lo mismo ocurre con PROGENES, que permite desarrollar diferentes bases de conocimiento sobre el mismo tema, correspondientes a distintos niveles de experiencia.

4.2 Activación del conocimiento

La selección del conocimiento pertinente a un problema dado es un aspecto fundamental a la hora de resolverlo. Un matemático no tiene en mente todas las definiciones y

teoremas que conoce cuando se enfrenta a un determinado problema, sino que empieza por considerar aquella parte de la materia que está directamente relacionada con el contexto en que se sitúa dicho problema, tratando de centrar la búsqueda de soluciones en el ámbito de conocimientos más restringido posible [POL65]. De esta forma, evita perder tiempo considerando propiedades que no tienen nada que ver con el problema en cuestión.

Del mismo modo, para evitar el consumo inútil de recursos que supone manejar un gran volumen de información innecesaria, PROGENES realiza una selección del conocimiento relevante para el problema que se va a resolver. La fase inicial de la resolución de un problema consiste en extraer del enunciado los conceptos clave que nos indican el contexto al que pertenece el problema, en función de lo cual se activan unas u otras bases de conocimiento. El resto del conocimiento permanece latente, inactivo, igual que en una persona.

Si el conocimiento que se está barajando se muestra insuficiente para resolver un problema, el matemático *relaja* la restricción del dominio, y empieza a considerar progresivamente áreas adyacentes a la considerada inicialmente [BAZ93b]. En estos casos, la resolución del problema se complica y puede llegar a ser realmente costosa. Sin embargo, este mecanismo es muy frecuente en la resolución de los problemas verdaderamente difíciles, que se salen del dominio en que se formulan en un principio. Conforme se profundiza en una disciplina científica, más difusas se vuelven las fronteras entre los distintos dominios. Por ejemplo, multitud de resultados de Teoría de Números se demuestran utilizando técnicas de otras áreas como Cálculo Diferencial y Variable Compleja.

Esto es bastante infrecuente en el tipo de problemas que pretendemos que PROGENES resuelva. No obstante, por generalidad, el sistema permite ampliar el ámbito de conocimiento utilizado a otras áreas próximas si es necesario: a medida que la búsqueda agota las posibilidades de deducción, se activan otras bases de conocimiento conocimiento próximas a las iniciales, de forma que se puede recurrir a técnicas propias de dominios que en principio no están directamente relacionadas con el problema.

4.3 Heurísticas para expresiones genéricas

Los matemáticos utilizan metaconocimiento de distinto nivel de generalidad. Por una parte, disponen de *know-how* aplicable a todo tipo de problema. También poseen metaconocimiento relacionado con áreas concretas de las Matemáticas: Geometría, Cálculo, Algebra, etc. Otros metaconocimientos más particulares se aplican en subdominios de las anteriores, como *superficies en* $\mathbb{R}^3$, *funciones reales*, o *álgebra matricial*, o en capítulos de éstos, como *integrales*, *geodésicas*, o *máximos y mínimos*. O incluso a un determinado grupo de problemas: *desigualdades entre integrales*, *comportamiento de una función real*, etc.

De igual manera, Prógenes incluye heurísticas y metareglas muy generales, y también otras más específicas a una determinada área, o a un tipo particular de problema. Entre las primeras se incluyen las que se describen en esta sección.

4.3.1 Proceduralización de la igualdad

En el sistema de Boyer y Moore se utilizan igualdades y equivalencias lógicas en forma de reglas de reescritura, asociando una direccionalidad a un conocimiento inicialmente declarativo. Pero este direccionamiento recae sobre el usuario, que es quien lo decide de antemano. De manera similar, la proceduralización y direccionamiento de las igualdades subyacentes a las metafunciones en PROGENES se establece al desarrollar la base de conocimiento, en base a la experiencia de un matemático.

El desarrollo de mecanismos que permitan que un sistema decida por sí mismo y de forma satisfactoria, en curso de resolución, que una igualdad debe ser direccionada, es una de las asignaturas pendientes en el campo de la demostración automática. También lo es la creación automática de funciones a partir de conocimiento declarativo. Como ya se apuntó en secciones precedentes, PROGENES no es capaz de definir metafunciones a menos que el usuario así lo especifique en la base de conocimiento, pero sí incluye, de forma limitada, la capacidad de proceduralizar conocimiento. El sistema decide en algunos casos que determinadas igualdades, que aparecen en forma declarativa en las hipótesis, se deben convertir en reglas de reescritura, de forma que se utilicen siempre para realizar substituciones automáticas e inmediatas en una dirección fija.

Este tipo de mecanismo es muy frecuente en la resolución de problemas por una

persona. Es lo que se hace, por ejemplo, en problemas de física en los que nos dan como dato algunas de las magnitudes que intervienen en el mismo. Si un problema nos dice que la velocidad de un móvil es $v(t) = \sin t$, podemos substituir $v(t)$ por $\sin t$ allí donde aparezca, y en cuanto aparezca, sin pensarlo ni considerar otra posibilidad, no sólo porque es correcto de acuerdo con las leyes de la lógica, sino porque también lo es desde el punto de vista *estratégico*, es decir, desde el punto de vista de lo que es conveniente para que podamos resolver el problema. Una vez hecha la substitución, la igualdad no nos es necesaria. Lo mismo haríamos con un hecho como $\lim_{t\to a} f(t) = 2$, o con las igualdades que dan valores explícitos para variables u objetos del problema, como $\alpha = \pi$, ó $radio(s) = 1$, donde s denota una esfera.[3]

En cambio, si sabemos que $\forall x, y \ f(x + y) = f(x).f(y)$, no está claro que sea conveniente substituir sistemáticamente $f(x+y)$ por $f(x).f(y)$, y menos aún olvidarnos de la igualdad anterior una vez hecha la substitución. Es probable que interese hacer algún tipo de manipulación sobre ella, o que en un momento dado haga falta hacer la substitución contraria. La pregunta es por tanto *¿cuál es el criterio según el cual una igualdad se puede tratar como regla de reescritura en una dirección determinada?*

PROGENES convierte una igualdad en regla de reescritura cuando uno de los miembros es *canónico salvo incógnitas*, es decir, canónico salvo por la aparición de variables libres del problema. El concepto de expresión canónica se explicó en el capítulo anterior, y viene a significar que la expresión representa un valor específico, explícito, en el que no hay nada por determinar. La idea es que hechos como $a = 5$, $f = \texttt{funcion\_real}\,(x, \tan x^2)$, $\lim_{t\to a} f(t) = 0$, ó $f = \texttt{funcion\_real}\,(x, px^2 + qx + 1)$, donde p y q son incógnitas, se convierten el reglas de reescritura de izquierda a derecha, mientras que $f_1 = g_1 + h_1$, donde f_1, g_1 y h_1 son constantes, ó $\forall x f(g(x)) = x$, permanecen en forma declarativa. Este último caso puede parecer un candidato a ser direccionado de izquierda a derecha. Sin embargo, en el segundo miembro, x, aparece una variable ligada exteriormente a la expresión, y por ello no es canónica salvo incógnitas.

Obsérvese que PROGENES no convertiría en regla de reescritura una igualdad como $\forall x \ f(x) = x^2$, cuando una persona haría la substitución de forma sistemática. En realidad PROGENES, como veremos más adelante, termina por crear una regla de reescritura

[3]Es posible que, si el valor conocido de una determinada expresión es una expresión muy complicada, una persona posponga la substitución hasta el último momento. Si por ejemplo sabemos que $x = \frac{\sqrt{1+2x^2}}{\log 2}$, es posible que la substitución de x por su valor complique las expresiones que tenemos que manejar, haciendo que el problema se vuelva confuso y farragoso. El ordenador tiene la ventaja de que puede manejar este tipo de expresiones sin dificultad, y por ello en PROGENES no supone ningún problema realizar la substitución desde el principio.

equivalente, ya que, como hemos visto ya en varios ejemplos, a partir de una igualdad universal como la anterior, infiere $f = \texttt{funcion\_real}\,(x, x^2)$, que sí se proceduraliza.

La idea de permitir que haya incógnitas en la expresión que va a substituir a la otra es que nos interesa conseguir el mayor número de hechos posible que involucren a los objetos que buscamos: cuanto más sepamos sobre ellos, más cerca estaremos de hallarlos.

La metaregla encargada de esta heurística es la siguiente:[4]

$$\boxed{y \text{ canonica salvo incognitas}} \quad \frac{\Gamma \vdash \varphi}{\Gamma, x = y \vdash \varphi} \quad \boxed{x := y}$$

En esta regla utilizamos la notación introducida al principio de esta sección, según la cual a la izquierda de la regla se pueden indicar metacondiciones adicionales para la aplicación de la misma, es decir, condiciones en el metalenguaje, que se evalúan proceduralmente, y a la derecha se indican acciones que se ejecutan antes de llevar a efecto el estado que se indica bajo la línea de la regla. De este modo, cuando se dispara esta metaregla, se crea la regla de reescritura $x := y$, que actúa de inmediato, substituyendo x por y en todo el problema. La igualdad $x = y$ se elimina de la base de hechos, ya que se convierte en $y = y$, que se reescribe a True.

PROGENES incorpora un segundo criterio para la proceduralización de igualdades, que consiste en que uno de los miembros corresponda a una metafunción de acceso a las componentes de un objeto, y el otro no contenga incógnitas. En tal caso, se crea una regla de reescritura que substituye aquél por éste, aunque éste no sea una expresión canónica. Es decir, si tenemos el hecho $\texttt{radio}(s_1) = r_1$, se crea la regla $\texttt{radio}(s_1) := r_1$. La metaregla que lo lleva a cabo es:

$$\boxed{\mathcal{F} \text{ funcion de acceso a slot}} \quad \frac{\Gamma \vdash \varphi}{\Gamma, \mathcal{F}(x) = y \vdash \varphi} \quad \boxed{\mathcal{F}(x) := y}$$

La idea es que cuando en un problema nos dan valores para las componentes de un objeto, nos están dando datos sobre el mismo, que podemos substituir en el problema.

[4]La consideración del caso simétrico $y = x$ viene garantizada por el tratamiento de la igualdad mediante clases de equivalencia (ver capítulo 3), combinado con un mecanismo incluido en el propio algoritmo de unificación, que considera todas las posibilidades de unificación entre dos igualdades, y genera todas las combinaciones posibles.

Las metafunciones de acceso a slot son por lo general conceptos secundarios que aportan una información relevante al problema, y no juegan un papel importante en las demostraciones. De esa forma, al eliminar la repetición de un mismo valor expresado de maneras distintas, se simplifican los problemas.

Por último, si dos constantes son iguales, entonces estamos designando un objeto por dos símbolos distintos, lo cual es innecesario, ya que un símbolo no es más que un nombre, y como expresión no aporta información alguna sobre el objeto que representa. Por tanto, las igualdades entre símbolos del item **CONSTANTES** se proceduralizan también.

Esta heurísticas simplifican enormemente las manipulaciones necesarias en los casos en que es posible aplicarla. Obviamente todo resultado que se obtenga por la aplicación de estas metareglas se puede obtener igualmente por substitución de igualdades, utilizando la regla de inferencia que dadas las hipótesis ψ y $x = y$, infiere el hecho $\psi_{[y/x]}$. Simplemente, las heurísticas de reescritura permiten hacerlo más rápido, y además *substituyendo* ψ por $\psi_{[y/x]}$ (en lugar de duplicar los hechos), sabiendo que hay garantías de que con toda probabilidad ψ no va a hacer falta, de manera que se limita el número de deducciones generadas por el sistema, llevando la demostración más rápidamente hacia los objetivos. La utilidad de este mecanismo se ha visto ya en los ejemplos mostrados en el capítulo anterior, y se ilustrará también en los problemas de este capítulo.

Como ya apuntamos en el capítulo anterior, las reglas de reescritura generadas en un subproblema a partir de igualdades de la base de datos local correspondiente, es decir, igualdades que son ciertas en el subproblema pero no globalmente, son inhibidas al abandonar el subproblema. Pensemos por ejemplo, en demostrar $(x = 0 \Rightarrow P) \wedge Q$. Empezamos por el primer término de la conjunción, en el que eliminamos la implicación, de forma que $x = 0$ se convierte en hecho, y al ser canónico el segundo miembro, se crea la regla de reescritura $x := 0$. Al pasar a demostrar Q, sería erróneo mantener esta regla, ya que $x = 0$ deja de ser cierto en cuanto se abandona el subobjetivo en el que se originó. El sistema mantiene en cada subestado de la demostración una lista de las reglas de reescrituras locales, de forma que se puedan desactivar al abandonar dicho estado.

4.3.2 Clase de equivalencia del objetivo

En el transcurso dé la resolución de un problema, pueden ir apareciendo diversos objetos y expresiones, y uno suele enfocar especialmente su atención sobre algunos de ellos. A menudo la consecución de la solución depende de que se haya uno fijado en los objetos adecuados. El sistema ONTIC [MCA89] utiliza un mecanismo de inferencia basada en un conjunto de *focus objects*, mediante el cual el sistema centra la deducción en determinados objetos que son clave para la resolución de un problema. El experto humano posee una cierta habilidad para reconocer qué elementos pueden ser importantes en un problema. Este mecanismo tiende a ser intuitivo y por tanto no es trivial para un programa, pues es difícil de formalizar. En ONTIC, es el usuario el que selecciona los *focus objects*, de forma que el sistema funciona mediante una ayuda externa.

En PROGENES, por el contrario, no se permite este tipo de ayuda por parte del usuario, ya que se pretende que la resolución de los problemas sea totalmente automatizada. PROGENES incluye un mecanismo que es menos general que el de ONTIC, pero que a cambio funciona sin ayuda externa. Se trata de una heurística que actúa en los casos en que el objetivo es una igualdad, y que otorga un lugar destacado a aquellos hechos que son igualdades en las que uno de los miembros unifica con uno de los miembros del objetivo. Cuando esto ocurre, el otro miembro del hecho se instancia con la substitución que resulta de la unificación, y se incluye en un item de la base de datos denominado **CLASE OBJETIVO**. También se meten en este item aquellos objetos que aparecen en una igualdad en la que el otro miembro unifica con alguno de los elementos del item. De esta forma, el item recoge la clase de equivalencia de todos los objetos que son iguales a (en realidad, que unifican con) los que aparecen en la igualdad a demostrar, o a otros que son iguales a éstos.

Una metaregla se encarga de hacer la selección en base al objetivo, y la otra en base al item **CLASE OBJETIVO**:

$$\boxed{\sigma \leftarrow unifica\,(x,t)} \quad \frac{\Gamma, x = y \underset{\Sigma, y_\sigma}{\vdash} t = u}{\Gamma, x = y \underset{\Sigma}{\vdash} t = u} \qquad \text{(I)}$$

$$\boxed{\sigma \leftarrow unifica\,(x,t)} \quad \frac{\Gamma, x = y \underset{\Sigma, t, y_\sigma}{\vdash} \varphi}{\Gamma, x = y \underset{\Sigma, t}{\vdash} \varphi} \qquad \text{(II)}$$

La función del sistema `unifica` devuelve la substitución de variables que hace que sus argumentos conicidan, o `False` en caso de que no exista tal substitución. La regla se aplica pues cuando la metacondición es distinta de `False`.

Las expresiones de esta clase de equivalencia se utilizan para generalizar determinadas heurísticas basadas en el objetivo, de forma que ciertas metareglas que utilizan información extraída del objetivo tienen una metaregla análoga, de menor prioridad, que hace lo propio con los elementos de este item, como veremos en el apartado 4.3.3 (de hecho la segunda de las metareglas anteriores es un ejemplo de ello). La idea es que toda información obtenida que involucre a estos objetos es *casi* tan útil en principio (*casi*, de ahí la menor prioridad) como la que se pueda obtener sobre un objeto buscado, o sobre uno de los dos miembros del objetivo, y puede por tanto conducir más o menos directamente a la consecución de éste. En el ejemplo 3 se ilustrará la utilidad de esta heurística.

También tiene sentido utilizar esta heurística respecto de un objeto buscado. Los problemas en que hay que demostrar una igualdad tienen muchas analogías con aquéllos en que hay que hallar un objeto.[5] Por ejemplo, un mecanismo típico para resolver este último tipo de problemas es la resolución de un sistema de ecuaciones. En ocasiones, este método funciona también para demostrar igualdades, como quedó patente en la resolución del problema 4, y como veremos en detalle más adelante. Inversamente, si buscamos un objeto x, y conseguimos demostrar una igualdad de la forma $x = expr$, y $expr$ es un valor admisible, tenemos la solución buscada.

Así pues, el item **CLASE OBJETIVO** incluye igualmente objetos del problema relacionados con un objeto buscado mediante una igualdad de los hechos, lo cual se refleja en la siguiente metaregla:

[5]Ambos problemas pueden coexistir, es decir, puede ocurrir que en un problema o en un estado de resolución haya un objetivo a demostrar y hechos buscados a la vez.

$$\boxed{\sigma \leftarrow unifica\,(x,t)} \quad \frac{\Gamma, x = y \overset{\Upsilon,t}{\underset{\Sigma,y_\sigma}{\vdash}} \varphi}{\Gamma, x = y \overset{\Upsilon,t}{\underset{\Sigma}{\vdash}} \varphi} \qquad \text{(III)}$$

Esto responde a la idea de que el hecho de que el objetivo sea de la forma $t = u$ es una situación similar (a estos efectos, equivalente) a aquélla en que se busca t.

4.3.3 Substitutividad de la igualdad

PROGENES incluye una serie de heurísticas basadas en el axioma de la lógica denominado de *substitutividad funcional* (ver p.e. [PAU87]), que controlan la aplicación del mismo en base al aspecto de determinados elementos del problema, como un hecho a demostrar o un objeto buscado. El axioma establece que para todo símbolo funcional f se tiene:

$$\forall x, y,\ x = y \Rightarrow f(x) = f(y) \qquad \text{(IV)}$$

donde x e y son términos cualesquiera del lenguaje.

En realidad el axioma se enuncia para funciones de aridad n arbitraria, pero en nuestro contexto nos basta con considerar el caso $n = 1$, ya que siempre substituiremos los argumentos de uno en uno, sin modificar los demás, lo cual equivale a *congelar* todos los argumentos menos uno, considerando la función de aridad 1:

$$f_{x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_n}(x) = f(x_1,\ldots,x_{j-1},x,x_{j+1},\cdots,x_n)$$

A partir del axioma se puede derivar la siguiente regla para el cálculo de *sequents*, que aplica esta propiedad de la igualdad en el contexto de un sistema de inferencia:[6]

[6]En un sistema de inferencia que incluya *modus ponens*, añadir esta regla de inferencia sería equivalente a incluir el axioma (III) en la base de reglas, de forma que se aplicaría por modus ponens. Si se escribe esta regla aquí es para poner de relieve la naturaleza de ciertos mecanismos que se describen más adelante.

$$\frac{\Gamma, x = y \vdash \varphi}{\Gamma, x = y, f(x) = f(y) \vdash \varphi} \qquad \text{(V)}$$

En Matemáticas esta regla se utiliza constantemente. La aplicación de una misma operación a los dos miembros de una igualdad es algo habitual en la resolución de problemas. Esto permite realizar manipulaciones como tomar logaritmos en los dos miembros de una igualdad, multiplicar por una misma matriz, derivar, integrar, tomar la inversa de funciones, números, matrices, etc. Sin embargo, los matemáticos no aplican este principio indiscriminadamente a cualquier igualdad que pueda aparecer en un problema, con cualquier operación posible. Lo aplican cuando creen que puede proporcionar información útil, de acuerdo con distintos criterios. Estos criterios no están fundados en argumentos teóricos, sino basados en la pragmática y el *know-how* sobre resolución de problemas.

De forma similar, la inclusión de una regla como (V) en un sistema de inferencia, sin agregarle algún tipo de mecanismo que controle su aplicación, sería muy desafortunada, ya que, siendo incontables las posibilidades de utilizarla, el sistema perdería una cantidad indefinida de tiempo infiriendo igualdades inútiles, e incluso podría entrar en un ciclo infinito del tipo $f(\cdots(f(x))\cdots) = f(\cdots(f(y))\cdots)$. Es necesario pues ejercer un control desde un nivel superior, desde el nivel en el que sabemos y observamos lo que estamos haciendo. PROGENES utiliza información de meta-nivel para decidir *cómo*, *dónde* y *cuándo* debe aplicarse esta propiedad de la igualdad. Como veremos en esta sección, esta información se refleja en una serie de metareglas formadas mediante la adición de metacondiciones a la regla (V).

Un problema puede dar indicios de que este tipo de manipulación nos puede acercar a la solución, y un experto en el área, o un estudiante aventajado, tienen la habilidad de detectar cuándo interesa aplicar el mecanismo, de forma que tienden a utilizarlo en situaciones específicas. Lo hacen, por ejemplo, al despejar una incógnita en una ecuación lineal, sumando, restando, multiplicando o dividiendo los dos miembros de la igualdad por un mismo término. En este caso, las operaciones se realizan mecánicamente y sin pensar, siguiendo una secuencia preestablecida (sumar términos de mismo orden, pasar el término independiente a la derecha cambiándole de signo, y pasar a la derecha el coeficiente de la x dividiendo). En casos como éste, en los que se conocen de antemano los pasos que resuelven el problema, PROGENES dispone de rutinas construidas al efecto, en las que la utilización del axioma está implícita. Por ejemplo, PROGENES dispone de metareglas que permiten detectar que un problema se puede resolver mediante un sistema de ecuaciones, en cuyo caso remite el sistema a

una rutina que lo resuelve.

Los matemáticos aplican también el axioma en casos más generales de resolución de ecuaciones que involucran transformaciones más complejas, como derivadas, logaritmos o integrales, para eliminar símbolos "molestos". Por ejemplo, para calcular la derivada en x de $y = f(x)^{g(x)}$ conociendo la derivada del logaritmo, se toman logaritmos en los dos miembros de la igualdad, de forma que se hace desaparecer el operador *potenciación*, tras lo cual se deriva y se despeja. Una manipulación similar permite calcular ciertos límites en los que aparecen logaritmos o exponenciales que nos estorban. PROGENES incluye metareglas para la eliminación de operadores en determinadas igualdades, como veremos más adelante.

Este tipo de manejo de la igualdad se utiliza también en situaciones en las que permite obtener ecuaciones que involucran objetos buscados, o igualdades que guardan una determinada semejanza con otra que hay que demostrar. Esto se consigue introduciendo los operadores apropiados que dan lugar a la similitud que nos interesa. Como veremos en el apartado que sigue, un operador se aplica a los dos miembros de una igualdad en la base de hechos si con ello se obtiene una nueva igualdad más próxima al objetivo a demostrar o a un objeto buscado.

Introducción de operadores

PROGENES incluye una heurística que actúa cuando el objetivo es una igualdad, buscando en las hipótesis igualdades que guarden una determinada similitud con aquél, o más concretamente, con alguno de sus miembros, en términos de que tanto en el objetivo como en la hipótesis aparezca, afectada por distintos operadores, una misma expresión. Dicho en términos intuitivos, se trata de que a un hecho *le falta* un operador para parecerse al objetivo. Dada esta situación, interesa aplicar a los dos miembros de la hipótesis los mismos términos que afectan a la subexpresión con la que coincide uno de sus miembros, y en el mismo orden en que aparecen.

A modo de introducción de esta heurística, veamos la resolución de uno de los problemas enunciados en el primer capítulo de esta tesis:

Problema 11. *Sea $f(x) = xg(x)$ para todo $x \in \mathbb{R}$. Sabiendo que g es continua en 0, demostrar que f es derivable en 0.*

Una vez reconocidas las distintas partes del problema y almacenadas en los items correspondientes, tenemos:

OBJETIVO : f_1 *derivable en* 0

HECHOS :

(1) $f_1, g_1 : \mathbb{R} \rightarrow \mathbb{R}$
(2) $\forall x \in \mathbb{R}\ f_1(x) = xg_1(x)$
(3) g_1 *continua en* 0

CONSTANTES :

f_1 : *funcion_real*
g_1 : *funcion_real*

Igual que en otros problemas anteriores, a partir del hecho (2), PROGENES añade la igualdad:

(4) $f_1 = funcion\_real(x, xg_1(x))$

que no se convierte en regla de reescritura debido a la presencia de la constante g_1 en el segundo miembro.

Entonces PROGENES, igual que haría probablemente un matemático, intenta sin éxito aplicar algunos teoremas para deducir nuevos hechos a partir de los que se conocen. También se intentan algunas heurísticas, como la de comprobar si f_1 es la composición de funciones y operadores derivables, pero todas fallan (por ejemplo, f_1 es el producto de dos funciones, pero no consta que g_1 sea derivable). De modo que el sistema aplica la definición de derivabilidad y continuidad en términos de límites por medio de la evaluación del código asociado a sendas metafunciones, de tal forma que el objetivo y el hecho (3) son substituidos por su definición. Entonces se aplica inmediatamente una heurística que se describirá en el apartado 4.4.3, que lleva a cabo un argumento matemático *built-in* que consiste en demostrar la existencia de un determinado límite buscando una igualdad que relacione la expresión que aparece en el límite con otra función que sepamos que tiene límite en el punto.

OBJETIVO : $funcion\_real(x, \frac{f_1(x) - f_1(0)}{x}) = g \ \wedge \ existe \lim_{x \rightarrow 0} g(x)$

INCOGNITAS : g : *funcion_real*

HECHOS :

...

(3') $\lim_{x \to 0} g_1(x) = g_1(0)$

...

Como efecto lateral de la presencia del concepto de límite en el hecho (3'), PROGENES da por supuesto que se cumple la condición correspondiente al concepto. En este caso, se añade el hecho:

(5) *existe* $\lim_{x \to 0} g_1(x)$

La conjunción del objetivo da lugar a sendos subproblemas, y el sistema empieza por demostrar el primero de ellos.

Ahora viene el punto clave de la resolución de este problema. Una persona podría darse cuenta de que f_1 aparece en el primer miembro de la igualdad que tenemos que demostrar, y también, despejada, en la igualdad (4), de forma que parece interesante aplicar a los dos miembros de (4) los mismos operadores que afectan a f_1 en el objeto buscado, pues de esta manera se obtiene una ecuación en la que éste aparece. La heurística para la introducción de operadores en igualdades, que vamos a explicar en este apartado, permite a PROGENES reproducir este mecanismo, cuya aplicación da lugar al hecho:

$$(6)\quad funcion\_real\,(x, \frac{f_1(x) - f_1(0)}{x}) = funcion\_real\,(x, \frac{funcion\_real\,(x, xg_1(x))\;(x) - funcion\_real\,(x, xg_1(x))\;(0)}{x})$$

Invocando la semántica procedural de aplicación inmediata asociada al concepto de imagen, y realizando automáticamente simplificaciones algebraicas elementales, obtenemos:

$$(6')\; funcion\_real\,(x, \frac{f_1(x) - f_1(0)}{x}) = funcion\_real\,(x, g_1(x))$$

Esta igualdad satisface el subobjetivo, con $g \rightarrow funcion\_real\,(x, g_1(x))$. Queda pues comprobar que *existe* $\lim_{x \to 0} funcion\_real\,(x, g_1(x))\;(x)$, que como consecuencia

de las propiedades procedurales de la metafunción imagen se convierte en *existe* $\lim_{x\to 0} g_1(x)$. El objetivo se cumple como consecuencia del hecho (5).

□

Obsérvese cómo la posibilidad de realizar cálculos simbólicos parece aquí un aspecto secundario, y sin embargo ha permitido que las heurísticas utilizadas en el problema llegasen a buen fin. En realidad sí son un aspecto trivial y secundario en el trabajo de un matemático. Los mecanismos de aplicación de conocimiento procedural de PROGENES permiten que también lo parezca así en las soluciones que obtiene.

El mecanismo que ha permitido introducir operadores como sumas, divisiones y construcción de un objeto de tipo `funcion_real` en la igualdad (4), obteniendo así el hecho (5'), está implementado en PROGENES por medio de la adición de condiciones de control a la regla de inferencia (V), de forma que su aplicación se restringe a casos más específicos, resultando metareglas como la siguiente:

$$\boxed{\sigma \leftarrow unifica\,(x,s)}\ \frac{\Gamma, x = y, (f(x) = f(y))_\sigma \vdash f(s) = u}{\Gamma, x = y \vdash f(s) = u} \qquad \text{(VI)}$$

Al aplicar esta regla, el sistema comprueba automáticamente que al aplicar f en los dos miembros de la igualdad se obtienen expresiones legales del lenguaje PROGENES. Esto significa que se deben verificar las condiciones que se establecen en la definición de los conceptos involucrados en la expresión (el chequeo de tipos es innecesario, ya que tiene lugar al hacer la unificación de x con s). Esto quiere decir, por ejemplo, que no podemos derivar funciones si no sabemos que son derivables.

Obsérvese el parecido de esta metaregla con la regla (V). La diferencia entre ambas reside en que mientras que (V) expresa conocimiento general correspondiente al dominio, en este caso, a la lógica, la regla (VI) especifica una configuración más concreta del estado del problema, junto con unas restricciones adicionales (indicadas a la izquierda de la regla).

En la práctica, en lugar del axioma (IV), PROGENES utiliza el teorema de *substitutividad de la igualdad en términos*, [PAU87], que generaliza a aquél, con la ventaja de que evita la necesidad de incluir una réplica de dicho axioma para cada símbolo funcional. El teorema dice:

$$\forall x, y,\ x = y \Rightarrow t_{[x/s]} = t_{[y/s]} \qquad \text{(VII)}$$

donde t, x, s e y son términos cualesquiera. Aunque el axioma (IV) se corresponde más directamente con la idea intuitiva que puede tener una persona de este tipo de tratamiento de la igualdad, este último teorema es más potente, flexible, y fácil de aplicar para el sistema. La construcción de una metaregla que pueda utilizarse para operadores de varios argumentos, basándose directamente en el axioma (IV), sería aparatosa y resultaría ineficiente. Por el contrario, esta formulación es independiente del número de argumentos de las funciones involucradas, y permite asimismo aplicar de una sola vez composiciones de operadores con un nivel de anidamiento arbitrario, lo cual es más apropiado para las características del lenguaje PROGENES. Como se indicó al principio de esta sección, los términos en el lenguaje PROGENES son las expresiones que denotan objetos, esto es, aquéllas cuya cabecera es una metafunción o un constructor.[7,8] Por ejemplo, esto permite que a partir de una igualdad como $\alpha + \beta = \gamma$ se deduzcan otras igualdades, como:

$$\texttt{distancia}(p,\ \texttt{plano}\ (2x + (\alpha + \beta)y - z + 2 = 0)) =$$
$$\texttt{distancia}(p,\ \texttt{plano}\ (2x + \gamma y - z + 2 = 0))$$

de una sóla vez y sin entrar en las complicaciones que plantea a la hora de la implementación la presencia de operadores que toman varios argumentos.

A este nivel es fundamental la distinción entre las funciones del lenguaje que estamos utilizando (el lenguaje PROGENES), es decir, metafunciones y constructores de objetos (operadores), y el concepto de función correspondiente al dominio que estamos tratando, y al que nos referimos como el tipo `funcion`. Este último es un concepto del lenguaje-objeto, mientras que las primeras son un concepto del metalenguaje. En Matemáticas muchos problemas se resuelven introduciendo no ya operadores predefinidos como sumas, logaritmos o límites, que corresponden a las metafunciones PROGENES, sino también funciones en el sentido de objetos PROGENES de tipo `funcion`. La

[7]Existe un teorema para predicados análogo al anterior, en el que la equivalencia lógica juega el mismo papel que tiene aquí la igualdad. Sin embargo, este teorema no se utiliza habitualmente en Matemáticas para añadir nuevas equivalencias $P \leftrightarrow Q$, sino más bien para añadir un hecho Q a partir de P, como se explicará en el apartado 4.3.5.

[8]Los objetos también se pueden denotar, como hemos visto, mediante variables tipadas, o también por constantes del lenguaje, como los números enteros. Este tipo de objetos también son términos del lenguaje, a los que se puede aplicar el teorema (VII).

notación que se utiliza en esta exposición para representar imágenes de funciones puede llevar a una confusión en este sentido. No debe olvidarse que $f(x)$ denota en este caso `imagen`(f,x), de modo que el símbolo funcional introducido es en realidad `imagen`, con un argumento fijo f. La formulación del principio de substitutividad mediante substitución de subexpresiones permite crear la ilusión de que realmente estamos aplicando f como símbolo funcional, tal como haríamos a nivel del dominio objeto. Con ello se acerca, se crea una equivalencia más directa entre el (meta)razonamiento que hace el sistema y el que seguiría un matemático cuando resuelve los mismos problemas.

La metaregla construida a partir de este teorema de forma análoga a lo visto anteriormente tiene la siguiente forma:[9,10]

$$\boxed{(s,\sigma)\leftarrow aparece\,(x,t)}\ \frac{\Gamma, x = y, (t = t_{[y/s]})_\sigma \vdash t = u}{\Gamma, x = y \vdash t = u} \qquad \text{(VIII)}$$

donde *aparece* es una función del sistema que comprueba si una expresión dada x unifica con alguna subexpresión de otra expresión t, en cuyo caso devuelve la subexpresión s que encuentra, junto con la substitución σ que resulta de la unificación. Por cada subexpresión encontrada, se crea una instanciación de la metaregla. En caso de no hallar ninguna, el valor devuelto es `False`. Obsérvese que la metaregla (VI) es un caso particular de ésta, en la que t es $f(s)$.

Igual que antes, para que la regla se aplique, $t_{[y/s]}$ debe ser una expresión bien construida, de modo que el sistema comprueba que se cumplen las condiciones asociadas a los conceptos involucrados en la expresión. Si estas condiciones no se cumplen, la regla no se aplica. Por ejemplo, en una expresión como $\lim_{x\to a} f(x)$ no podemos substituir f por g a menos que sepamos que el límite de g existe.

La regla (VIII) viene a expresar que cuando el objetivo es una igualdad, y hay un hecho que es también una igualdad, que tiene algo en común con el objetivo en cuanto a que uno de los miembros de aquélla *aparece* (módulo unificación) en uno de los miembros de ésta, se crea un hecho nuevo a partir del existente, que tenga aún más en común con el objetivo, concretamente, un miembro en común (salvo instanciación). Así por ejemplo, si tenemos que demostrar $f(a) = expr_1$, y sabemos que $a = b$, añadiríamos

[9] Obsérvese que el miembro izquierdo de la nueva igualdad que se añade a las hipótesis es t, y no $t_{[x/s]}$. En este caso, la substitución en t es superflua, ya que $(x = s)_\sigma$.

[10] Recuérdese que la consideración de los casos simétricos (es decir, $u = t$, $y = x$) viene garantizada por el tratamiento de la igualdad mediante clases de equivalencia en el motor de inferencia.

el hecho $f(a) = f(b)$, que aporta una información directamente relacionada con el (cuando no igual al) objetivo. En este ejemplo, la manipulación realizada equivale a introducir f en los dos miembros de la igualdad $a = b$, aunque, como se explicó antes, la función introducida es en realidad el operador imagen.

Esta metaregla es de alta prioridad, ya que se aplica en casos bastante concretos, y sus efectos son casi siempre de gran utilidad en las demostraciones.

Obsérvese que en la metaregla no se exige estrictamente que x aparezca en t, es decir, que sea una subexpresión de t, sino que se expresa una condición más *débil*: que alguna subexpresión de t unifique con x. Para ilustrar las posibilidades que esto permite, consideremos un caso sencillo:

Problema 5. *Demostrar que si g es una función par entonces para cualquier función real f se tiene que $f \circ g$ es par.*

OBJETIVO : $\forall f : funcion\_real,\ f \circ g_1\ par$

HECHOS : (1) $g_1\ par$

CONSTANTES : $g_1 : funcion\_real$

Eliminando la cuantificación del objetivo tenemos:

OBJETIVO : $f_1 \circ g_1\ par$

CONSTANTES :

...

$f_1 : funcion\_real$

Igual que en ejemplos anteriores, al no haber teoremas que se puedan utilizar, se aplica la definición de función par, así como la de imagen de una composición. Tras eliminar el cuantificador $\forall$ del objetivo, tenemos:

OBJETIVO : $f_1(g_1(z_1)) = f_1(g_1(-z_1))$

HECHOS :

(1') $\forall z \in \mathcal{D}_{g_1},\ g_1(z) = g_1(-z)$
(2) $z_1 \in \mathcal{D}_{g_1}$

CONSTANTES :

...
$z_1 \in \mathbb{R}$ [11]

Obsérvese que en el hecho (2) se ha simplificado $\mathcal{D}_{f_1 \circ g_1}$ a $\mathcal{D}_{g_1}$, utilizando propiedades procedurales de aplicación automática de la metafunción `dominio`.

Puesto que el miembro izquierdo de la hipótesis (1'), $g_1(z)$, unifica con una subexpresión, $g_1(z_1)$, del miembro izquierdo del objetivo mediante la substitución $\sigma \rightarrow [z_1/z]$, se aplica la metaregla (VIII), con la instanciación $[g_1(z)/x, g_1(-z)/y, f_1(g_1(z_1))/t, f_1(g_1(-z_1))/u]$, y $s \rightarrow g_1(z_1)$, de forma que se deduce $f_1(g_1(z_1)) = f_1(g_1(-z_1))$, que es justamente lo que buscábamos.[12]

□

Este problema se podría haber resuelto también por substitución de $g(z_1)$ por $g(-z_1)$ (o viceversa) en el objetivo, aplicando la igualdad que aparece en la hipótesis (1'). Sin embargo, la heurística que se ha utilizado es perfectamente válida, y además necesaria en otros problemas que no se pueden resolver por meras substituciones. En el apartado 4.3.5 se dará un mecanismo heurístico de substitución de igualdades en el objetivo y se discutirá su relación con la heurística que se describe en el presente apartado.

En sistemas como MUSCADET, se evitan los anidamientos en las expresiones mediante la eliminación de los símbolos funcionales, introduciendo nuevas variables. Así, si en un problema aparece una expresión como $f(g(x))$, se introduce una variable z, se añade la igualdad $z = g(x)$, y se cambia la expresión anterior por $f(z)$, en la que f se elimina por el mismo procedimiento. Este mecanismo recibe el nombre de *flattening*. En PROGENES por el contrario, no se eliminan los anidamientos de símbolos funcionales, gracias a lo cual, en particular, la subexpresión s, en la metaregla (VIII), puede tener un nivel de anidamiento arbitrario dentro de la expresión t. Esto hace que la regla resulte

[11] Recuérdese que $x \in \mathbb{R}$ es una abreviatura de `x:real`.

[12] Observando los miembros derechos de las igualdades, la metaregla se puede aplicar también, mediante la instanciación $[g_1(-z)/x, g_1(z)/y, f_1(g_1(-z_1))/t, f_1(g_1(z_1))/u]$, con $s \rightarrow g_1(-z_1)$, y $\sigma \rightarrow [z_1/z]$, obteniendo exactamente el mismo resultado.

potente y al mismo tiempo sencilla de escribir. De esta forma el sistema permite una mayor riqueza expresiva para representar conocimiento y metaconocimiento.

Por otro lado, en particular es posible también que la subexpresión s de t sea el mismo t, es decir, que x unifique directamente con t. En este caso, el operador a aplicar en la igualdad $x = y$ sería la identidad, o lo que es lo mismo, no se aplicaría ningún operador, de forma que el hecho que se añadiría es $(x = y)_\sigma$, lo que significa que en realidad se está haciendo sencillamente una instanciación de un hecho. Esta forma de aplicar la metaregla permite así realizar instanciaciones útiles o "interesantes" de hechos que contienen variables libres o cuantificadas universalmente, puesto que dan lugar a expresiones que se parecen al objetivo. Esta forma de aplicación de la heurística se ilustrará en varios de los problemas del final de este capítulo.

¿Qué ocurre si en vez de tener que demostrar $f(a) = f(b)$ como la anterior tenemos que hallar un objeto como $f(a)$, sabiendo que $a = b$? Deberíamos ser capaces de realizar la misma operación que antes, ya que $f(a) = f(b)$ también es una información interesante de cara a determinar $f(a)$. PROGENES incluye otra metaregla análoga a la anterior, que en lugar de basarse en el objetivo para controlar la aplicación de la propiedad, hace lo propio con un objeto buscado:

$$\boxed{(s,\sigma) \leftarrow aparece\,(x,t)} \quad \frac{\Gamma, x = y, (t = t_{[y/s]})_\sigma \overset{\Sigma,t}{\vdash} \varphi}{\Gamma, x = y \overset{\Sigma,t}{\vdash} \varphi} \qquad \text{(IX)}$$

Aquí el objeto buscado desempeña el mismo papel que juega uno de los miembros del objetivo en la metaregla (VIII), de forma que todo lo dicho para la otra regla es válido también para ésta. Esta metaregla es la que permite resolver problemas como el mostrado al principio de este apartado, o como el problema 4, resuelto en el primer capítulo. Más adelante veremos más casos en los que se utiliza.

En principio el elemento de referencia legítimo en este tipo de razonamiento debe ser, como hemos visto hasta aquí, el resultado (objetivo u objeto) a obtener. No obstante, hay otros elementos del problema, que sin ser tan centrales como éstos, también merecen una atención especial, precisamente por su relación con ellos. Como vimos en el apartado 4.3.2, el sistema concede una atención especial a los hechos y objetos que tienen una conexión directa con el hecho a demostrar o con un objeto buscado, utilizando a tal efecto el item **CLASE OBJETIVO**. Los elementos de este item se utilizan para guiar este mecanismo, ya que cualquier información obtenida que

tenga relación con ellos, con toda probabilidad la tendrá también con el objetivo o los objetos buscados según el caso. PROGENES incluye una metaregla al efecto, análoga a las dos anteriores, aunque de prioridad más baja:

$$\boxed{(s,\sigma)\leftarrow aparece\,(x,t)}\ \frac{\Gamma, x = y, (t = t_{[y/s]})_\sigma \underset{\Sigma,t}{\vdash} \varphi}{\Gamma, x = y \underset{\Sigma,t}{\vdash} \varphi} \qquad \text{(X)}$$

Las metareglas (VIII), (IX) y (X) son análogas en cuanto a intención. Todas ellas aplican heurísticas para controlar la aplicación del teorema (VII). Lo que varía de unas a otras es el elemento de atención en función del cual se aplica la inferencia. En (VIII) es el objetivo, en (IX) un objeto buscado, y en (X) una expresión estrechamente relacionada con alguno de los elementos anteriores. La regla (X) se utiliza con menos frecuencia que las otras dos, pero es necesaria para resolver ciertos tipos de problemas, como veremos más adelante.

Las metareglas de introducción de operadores han demostrado ser un método muy potente y de amplia utilidad, que responde a un mecanismo típico, absolutamente natural, y muy frecuente en la resolución de problemas de diversa índole. Esta técnica permite al sistema resolver un gran número de problemas de manera sencilla y eficaz, como hemos visto en los ejemplos, y veremos en otros problemas.

Eliminación de operadores

Otra situación que sugiere la utilización del axioma de substitutividad funcional de la igualdad es la situación contraria a la que se ha descrito en el apartado anterior, es decir, aquélla en la que nos estorba un operador en una igualdad de los hechos. Es el objetivo (caso *hallar*) o uno de sus miembros (caso *demostrar* una igualdad) el que aparece como subexpresión en uno de los miembros de un hecho que es una igualdad. En este caso, interesa despejar dicha subexpresión, es decir, aplicar la inversa del operador que aparece en la expresión que la contiene, para eliminarlo. Obviamente, una condición previa para que esta idea se pueda aplicar es que dicho operador tenga una inversa. Esta heurística se plasma en metareglas como la siguiente:

$$\boxed{\textit{inversa}(g,f),\ \sigma\leftarrow\textit{comun}(x,t)}\ \frac{\Gamma, f(t)=u, (g(f(t))=g(u))_\sigma \vdash x=y}{\Gamma, f(t)=u \vdash x=y}$$

donde el predicado *comun* verifica si dos expresiones tienen alguna subexpresión en común, es decir, dos subexpresiones que unifiquen. Se trata de que no sólo se pueda aplicar la regla cuando x aparece en t, sino también cuando aparece en t alguna subexpresión de x, de forma que si conseguimos despejar esta subexpresión, después podremos introducir los operadores que haga falta hasta obtener x. Esta combinación de la metaregla anterior con la de introducción de operadores se ilustrará mediante un ejemplo al final de este apartado.

La base de conocimiento incluye una serie de hechos que expresan la información necesaria para utilizar esta regla, como `inversa(integral, derivada)`, `inversa(sqr, sqrt)`, ó `inversa(exp, log)`. Algunos operadores son inversos en una dirección, pero no en la otra, como es el caso de la derivada y la integral, pero el razonamiento se puede realizar a pesar de todo.

A diferencia de la introducción de operadores, esta heurística se define a partir del axioma (IV) de substitutividad funcional, y no del teorema que lo generaliza, ya que esta manipulación se basa en propiedades del operador concreto que aparece, por lo que es necesaria la aparición explícita del mismo, mientras que la heurística anterior es independiente de los operadores que intervienen en las expresiones.

Igual que en la introducción de operadores, existe una metaregla análoga a la anterior, que permite un razonamiento similar a partir de un objeto buscado:

$$\boxed{\textit{inversa}(g,f),\ \sigma\leftarrow\textit{comun}(x,t)}\ \frac{\Gamma, f(t)=u, (g(f(t))=g(u))_\sigma \overset{\Sigma,x}{\vdash} \varphi}{\Gamma, f(t)=u \overset{\Sigma,x}{\vdash} \varphi}$$

Asimismo, existen sendas reglas de menor prioridad análogas a las dos anteriores basadas en los objetos del item **CLASE OBJETIVO**, en lugar del objetivo.

Este tipo de manipulación de igualdades permite eliminar operadores que estorban, para despejar incógnitas, o para obtener un cierto resultado. Así por ejemplo, si tenemos que hallar una función f, y sabemos que $\int_0^{x^3} f(t)dt = \cos x$, aplicaremos la derivada respecto de x a los dos miembros de la igualdad. Posteriormente habrá que seguir manipulando la ecuación hasta obtener una expresión del tipo $f(x) = \cdots$.

La combinación de las metareglas de eliminación con las de introducción de operadores ofrece posibilidades interesantes. Para mostrarlo, veamos la solución que proporciona PROGENES para el problema 14:

> **Problema 14.** *Sea* $f : \mathbb{R} \to \mathbb{R}$ *tal que* f' *es periódica de periodo* p. *Demostrar que* f *es también periódica de periodo* p *si y sólo si* $f(p) = f(0)$.

La doble implicación en el objetivo da lugar a una conjunción que se descompone en dos subproblemas. PROGENES empieza por resolver:

OBJETIVO : $f_1(p_1) = f_1(0) \Rightarrow f_1\ p_1\text{−}periodica$

HECHOS :

(1) $f_1 : \mathbb{R} \to \mathbb{R}$
(2) $f_1'\ p_1\text{−}periodica$

CONSTANTES :

$f_1 : funcion\_real$
$p_1 \in \mathbb{R}$

Nada más leer el enunciado, PROGENES añade la constante atómica 0 a la lista de constantes:

CONSTANTES : $\ldots, 0$

Esto se hace sistemáticamente, aunque en otros problemas que se muestran en esta tesis se omite este aspecto cuando no tiene relevancia para la resolución de los mismos.

Para demostrar una implicación, se asume la parte izquierda. Al no haber reglas que puedan aplicarse, se ejecuta la definición de la metafunción `periodica`, y se elimina la cuantificación universal que aparece con ello en el objetivo, tras lo cual tenemos:

OBJETIVO : $f_1(x_1) = f_1(x_1 + p_1)$

HECHOS :

...

(2') $\forall x \in \mathbb{R},\ f_1'(x) = f_1'(x + p_1)$

(3) $f_1(p_1) = f_1(0)$

CONSTANTES :

...

$x_1 \in \mathbb{R}$

Este problema no se puede resolver por medio de técnicas de resolución de ecuaciones diferenciales, debido a que aparece el término $f_1'(x + p_1)$.

A partir de la igualdad (2'), se añade otra semejante en términos de funciones (que no se proceduraliza debido a la presencia de constantes en el miembro derecho):

(4) $f_1' = funcion\_real\,(x, f_1'(x + p_1))$

Ahora la metaregla descrita en este apartado se da cuenta de que el primer miembro de (4), f_1', tiene una subexpresión, f_1, en común con el primer miembro del objetivo, $f_1(x_1)$, de modo que se aplica la inversa de la derivada, es decir, se integra, en (4):

(5) $\int f_1' = \int funcion\_real\,(x, f_1'(x + p_1))$

La semántica procedural de la metafunción `integral` se evalúa en este caso automáticamente, de forma que se obtiene:

(5') $f_1 = funcion\_real\,(x, f_1(x + p_1) + k_1)$

donde k_1 es una constante que resulta de calcular las primitivas:[13]

CONSTANTES :

...

$k_1 \in \mathbb{R}$

El miembro izquierdo de (5') aparece en el primer miembro del objetivo, de forma que se aplica la regla de introducción de operadores (caso identidad), instanciando (5') con $[x_1/x]$:

[13]En realidad, se crea el hecho $\exists k \in \mathbb{R}$ *tal que* $f_1 = funcion\_real\,(x, f_1(x + p_1) + k)$, en el que se elimina la cuantificación existencial, creando la constante k_1.

(6) $f_1(x_1) = f_1(x_1 + p_1) + k_1$

Por otro lado, a partir de (5'), que es una igualdad entre funciones, el sistema añade la correspondiente igualdad entre imágenes:

(7) $\forall x \in \mathbb{R},\ f_1(x) = f_1(x + p_1) + k_1$

Este hecho universal se instancia con las constantes del problema. Sólo mostraremos aquí la instanciación $[0/x]$, que es la que se necesita para la resolución del problema (el hecho (2), también se instancia, pero no interviene en la resolución, por lo cual se omite aquí.):

(8) $f_1(0) = f_1(p_1) + k_1$

Ahora tenemos que (3), (6), y (8) forman un sistema de ecuaciones que involucra a uno de los miembros del objetivo, el primero, y que se pueden resolver, y cuya solución $f_1(x_1) = f_1(x_i + p_1)$, es lo que buscábamos demostrar.

Falta demostrar la implicación en sentido contrario, que es un problema trivial:

OBJETIVO : $f_1\ p_1\text{−}periodica \Rightarrow f_1(p_1) = f_1(0)$

HECHOS :

(1) $f_1 : \mathbb{R} \rightarrow \mathbb{R}$
(2) $f_1'\ p_1\text{−}periodica$

Eliminando la implicación del objetivo y aplicando la definición de periodicidad de funciones, tenemos:

OBJETIVO : $f_1(p_1) = f_1(0)$

HECHOS :

...
(2') $\forall x \in \mathbb{R},\ f_1'(x) = f_1'(x + p_1)$
(3) $\forall x \in \mathbb{R},\ f_1(x) = f_1(x + p_1)$

Puesto que el primer miembro de (3) unifica con el segundo del objetivo, la regla de introducción de operadores (caso identidad) instancia la hipótesis con $[0/x]$, obteniendo

justamente el hecho buscado (la unificación con el primer miembro también es posible, dando lugar a $[p_1/x]$, que no conduce a nada).

□

4.3.4 Generalización de la introducción de operadores

El tipo de razonamiento descrito en los apartados anteriores no es exclusivo de las igualdades. Considérese el siguiente problema:

> **Problema 8.** *Sabiendo que* $x+1 \leq f(x) \leq e^x$ *para todo* x*, demostrar que* f *es continua en el origen.*

OBJETIVO : f_1 *continua en* 0

HECHOS :

(1) $f_1 : \mathbb{R} \rightarrow \mathbb{R}$
(2) $\forall x \in \mathbb{R},\ x+1 \leq f_1(x)$
(3) $\forall x \in \mathbb{R},\ f_1(x) \leq e^x$

Tras fracasar los intentos de resolver el problema utilizando teoremas, se aplica en el objetivo la definición del metapredicado `continua`, de manera que obtenemos:

OBJETIVO : $\lim_{t \to 0} f_1(t) = f_1(0)$

Al aparecer el concepto de `limite` en el objetivo, el sistema añade como subobjetivo la condición asociada a la metafunción, que en este caso supone demostrar que el límite existe. Antes de utilizar la definición de límite en términos de ε y δ, el sistema aplica en backward chaining un teorema que establece:

$$\underline{\lim_{t \to a}} f(t) = \overline{\lim_{t \to a}} f(t) \implies \textit{existe } \lim_{t \to a} f(t) \ \wedge \ \underline{\lim_{t \to a}} f(t) = \overline{\lim_{t \to a}} f(t)$$

De forma que se plantea el subobjetivo:

OBJETIVO : $\underline{\lim_{t \to 0}} f_1(t) = \overline{\lim_{t \to 0}} f_1(t)$

El primer miembro de (3) y el segundo de (2) unifican con una subexpresión del primer miembro del objetivo, $f_1(t)$, mediante la substitución $[t/x]$. Si (2) y (3) fueran igualdades, se podría aplicar la regla de introducción de operadores (VIII) para tomar límites en los dos miembros de estos hechos. Aunque en este caso los hechos son desigualdades, parece que esta manipulación puede ser aconsejable, ya que la información que proporciona, aún no siendo tan determinante como una igualdad, está directamente relacionada con el objetivo. En este apartado se describirá una heurística análoga a la de introducción de operadores, pero para desigualdades en vez de igualdades, que permite a PROGENES asertar a partir de (2) y (3) los hechos:

(4) $\underline{\lim}_{t\to 0}\,(t+1) \leq \underline{\lim}_{t\to 0}\, f_1(t)$

(5) $\underline{\lim}_{t\to 0}\, f_1(t) \leq \underline{\lim}_{t\to 0}\, e^t$

Puesto que el primer miembro de (3) y el segundo de (2) también aparecen en el segundo miembro del objetivo, se deducen igualmente los hechos:

(6) $\overline{\lim}_{t\to 0}\,(t+1) \leq \overline{\lim}_{t\to 0}\, f_1(t)$

(7) $\overline{\lim}_{t\to 0}\, f_1(t) \leq \overline{\lim}_{t\to 0}\, e^t$

A primera vista uno puede pensar que esta heurística se podría haber aplicado ya en un primer momento, antes de hacer backward chaining, introduciendo en (2) y (3) el operador `limite` que aparece en el objetivo inicial. Sin embargo esto no es posible, ya que no podemos tomar límites sin saber que existen. En cambio los límites inferior y superior siempre existen.

Las rutinas de cálculo asociadas a los operadores `limite_inferior` y `limite_superior` dan lugar a:

(4') $1 \leq \underline{\lim}_{t\to 0}\, f_1(t)$

(5') $\underline{\lim}_{t\to 0}\, f_1(t) \leq 1$

(6') $1 \leq \overline{\lim}_{t\to 0}\, f_1(t)$

(7') $\overline{\lim}_{t\to 0}\, f_1(t) \leq 1$

Inmediatamente después, se aplica la regla de producción:

$$\forall x, y,\ x \mathcal{R}_1 y \Rightarrow \mathcal{F}(x)\ \mathcal{R}_2\ \mathcal{F}(y)$$

Donde $\mathcal{R}_1$ y $\mathcal{R}_2$ son en principio desigualdades, aunque podrían ser cualquier otra relación, como comentaremos más adelante. Obsérvese que este tipo de propiedad es similar al teorema de substitutividad de la igualdad. En el caso en que $\mathcal{R}_1$ y $\mathcal{R}_2$ fueran ambas la igualdad, todo operador $\mathcal{F}$ verificaría la propiedad anterior.

Así pues, de la idea de *aplicar la misma función a los dos miembros de una igualdad* pasamos a un principio más general: *aplicar una misma transformación a los argumentos de una determinada relación.*

Igual que ocurría con el teorema (IV) descrito más atrás, si incluyésemos en forma de regla este tipo de propiedades de los operadores, se provocaría una tendencia al crecimiento excesivo e indiscriminado de la base de hechos. Además, si $\mathcal{R}_1$ y $\mathcal{R}_2$ son la misma relación, la regla se hace cíclica, ya que se puede aplicar a los hechos que ella misma produce, pudiendo entrar en un ciclo infinito, en el que se infieren indefinidamente hechos de la forma $\mathcal{F}(\cdots(\mathcal{F}(x))\cdots)\ \mathcal{R}_1\ \mathcal{F}(\cdots(\mathcal{F}(y))\cdots)$. Es por ello que se hace necesario una vez más un mecanismo que controle la aplicación de este conocimiento. El criterio utilizado para la introducción de operadores en igualdades es perfectamente válido en este caso, matizado por condiciones adicionales, ya que no se puede introducir cualquier operador. La intención sigue siendo la misma: partiendo de un hecho conocido, operar sobre él, introduciendo elementos que hacen que se parezca cada vez más al objetivo, ocasionando ligeras modificaciones, hasta obtener el hecho que se buscaba demostrar. Es decir, la aplicación de una propiedad del dominio viene condicionada por el aspecto del objetivo a demostrar. Análogamente al caso de la igualdad, el razonamiento se puede hacer aquí también en función de un objeto buscado, tratando de obtener una igualdad que nos proporcione el valor deseado.

Así pues, PROGENES incluye una heurística que permite deducir nuevas desigualdades a base de realizar una misma operación en los dos miembros de otras, en función del resultado que busquemos obtener (un hecho a demostrar o un objeto buscado), y en base al conocimiento que tiene el sistema sobre las propiedades de determinados operadores respecto de las desigualdades, conocimiento que establece cuál es la relación resultante de la aplicación del operador a cada tipo de desigualdad. Esta heurística se

implementa mediante la siguiente metaregla:[14,15,16]

$$\boxed{\sigma \leftarrow aparece(\mathcal{F}(a),\varphi)}\ \frac{\Gamma,\ [\forall x \forall y\ x\mathcal{R}_1 y \Rightarrow \mathcal{F}(x)\ \mathcal{R}_2\ \mathcal{F}(y)],\ a\mathcal{R}_1 b,\ (\mathcal{F}(a)\ \mathcal{R}_2\ \mathcal{F}(b))_\sigma\ \vdash\ \varphi}{\Gamma,\ [\forall x \forall y\ x\mathcal{R}_1 y \Rightarrow \mathcal{F}(x)\ \mathcal{R}_2\ \mathcal{F}(y)],\ a\mathcal{R}_1 b\ \vdash\ \varphi} \qquad \text{(XI)}$$

Igual que para la introducción de operadores en igualdades, este tipo de inferencia sólo se puede llevar a cabo cuando las expresiones construidas están bien definidas de acuerdo con las condiciones que aconmpañan a la definición de los conceptos involucrados.

Obsérvese que, a diferencia de las reglas de introducción de operadores en igualdades, ésta no se expresa en términos de substitución en expresiones, sino que el operador a introducir aparece explícitamente. Esto se debe a que el operador debe cumplir ciertos requisitos ($x\mathcal{R}_1 y \Rightarrow \mathcal{F}(x)\ \mathcal{R}_2\ \mathcal{F}(y)$), por lo cual es necesaria una referencia explícita al mismo en la regla. Una de las consecuencias de esto es que es necesario escribir una segunda metaregla para el caso en que, como en el ejemplo del principio de este apartado, se quiere realizar instanciaciones de $a\mathcal{R}_1 b$ sin introducir ningún operador (introduciendo la identidad):[17]

$$\boxed{\sigma \leftarrow aparece(a,\varphi)}\ \frac{\Gamma, a\mathcal{R}_1 b, (a\mathcal{R}_1 b)_\sigma,\ \vdash\ \varphi}{\Gamma, a\mathcal{R}_1 b\ \vdash\ \varphi}$$

Igual que con la heurística de introducción de operadores en igualdades, sendas metareglas análogas a éstas permiten aplicar el mismo mecanismo cuando $\mathcal{F}(a)$ aparece en un objeto buscado. Es posible extender asimismo el mecanismo para que trabaje con objetos del item **CLASE OBJETIVO**, aunque no nos han surgido hasta el momento ejemplos en los que esto sea necesario.

Aunque en la metaregla (XI) se expresa la presencia entre las hipótesis de la propiedad de conservación en forma de implicación, en la práctica este hecho se ex-

[14] Aquí también *aparece* significa que una expresión unifica con una subexpresión de otra.

[15] A diferencia de la regla de introducción de operadores en igualdades, en este caso, $\mathcal{F}$ sólo puede ser una metafunción, ya que a los constructores de objetos no les corresponden propiedades de preservación de relaciones de orden, ni en PROGENES, ni en el conocimiento humano.

[16] El sistema incluye una metaregla análoga para el caso en que *aparece* $(\mathcal{F}(b), \varphi)$.

[17] Igual que antes, otra metaregla considera el caso *aparece*(b, φ), ya que estas relaciones no son simétricas.

presa mediante una sintaxis específica, como veremos más adelante, evitando así que el sistema la utilice como regla de producción que se aplicaría sin restricciones mediante *modus ponens*, que es justamente lo que se pretende evitar.

La regla (XI) se podría restringir más aún, exigiendo que φ sea una relación binaria *próxima* a $\mathcal{R}_2$, es decir, que si por ejemplo ésta última es una desigualdad, aquélla también lo sea, aunque de otro tipo, de tal forma que de aquí se deduzcan hechos que apunten directamente a la obtención del objetivo. De los ejemplos que hemos tratado no se desprende que la restricción de la regla sea o no conveniente en general. En principio hemos optado por la versión más general, considerando que vale la pena dedicir hechos que aportan información relacionada con subexpresiones del objetivo, aunque no conduzcan directamente a éste. No obstante, la opción restrictiva es igualmente válida, y para aplicarla basta con añadir a la metaregla la metacondición correspondiente.

Aunque inicialmente concebidas para tratar desigualdades, por la forma en que están escritas, estas metareglas se pueden aplicar a otras relaciones binarias. De hecho, la idea de un operador que preserva determinada relación, o la transforma en otra más o menos próxima, es bastante común en Matemáticas. Por ejemplo, la intersección de conjuntos, la unión, la imagen mediante una función y la imagen inversa mediante una función inyectiva, preservan la relación de inclusión entre conjuntos, mientras que la complementación la invierte. Las relaciones *'ortogonal'* y *'tener igual norma'* en espacios vectoriales se mantienen bajo isometrías, la relación unaria *'ser abierto'* en espacios topológicos se conserva bajo la imagen inversa de una función siempre que ésta sea continua, y la composición de funciones mantiene la continuidad y la inyectividad. Dado que hasta el momento PROGENES se ha aplicado fundamentalmente a problemas de Análisis Matemático y Geometría Analítica, este mecanismo en la práctica se utiliza básicamente para el caso particular de las desigualdades, es decir cuando $\mathcal{R}_1$ y $\mathcal{R}_2$ son $<$, $\leq$, $>$, ó $\geq$. Creemos sin embargo que puede ser interesante estudiar la utilidad de este método en dominios como los citados más arriba, o en otros distintos, considerando incluso la generalización del mecanismo a relaciones n-arias.

El tratamiento de la substitutividad de la igualdad se podría haber incluido como parte de esta heurística, ya que es un caso particular (en el que $\mathcal{R}_1$ es la igualdad). Sin embargo, la heurística específica para la igualdad descrita en el apartado 4.3.3 ha demostrado tener un rango de utilidad particularmente amplio, con diferencia mayor que el del caso general. Esto, junto con el hecho de que es innecesario comprobar que un operador preserva la igualdad, lo cual permite una formulación más simple y directa de las metareglas, hace que resulte preferible, por más eficiente, tratar por separado el caso de la igualdad, añadiendo a la metaregla general la condición de que $\mathcal{R}_1$ no

sea la igualdad, para evitar redundancias. Además al no depender el mecanismo del operador que se introduzca, no es necesario hacer referencia explícita a él en la regla, resultando así una formulación más sencilla de la heurística, basada en substituciones en expresiones arbitrarias que denotan términos. Esto permite en particular que las metareglas de introducción de operadores hagan en un sólo paso lo que la metaregla general haría en varios. Por ejemplo, si el objetivo es demostrar $g(f(x,y),h(z)) = u$, y tenemos el hecho $x = v$, aplicando la metaregla general, primero deduciríamos $f(x,y) = f(v,y)$, y después $g(f(x,y),h(z)) = g(f(v,y),h(z))$. Con la regla de la igualdad, obtendríamos este hecho de una sola vez.

Por otra parte, igual que hemos generalizado la introducción de operadores en igualdades al caso de inecuaciones u otro tipo de relaciones, se podría considerar una generalización similar de la eliminación de operadores. No se ha experimentado con esta posibilidad, pero podría tener utilidad en determinado tipo de problemas.

Con el objeto de facilitar la comprensión del mecanismo, en este apartado se se ha presentado una versión de la metaregla (XI) que actúa con metafunciones de aridad 1. En la práctica, el sistema maneja una versión en la que se admiten metafunciones de aridad arbitraria, como `imagen`, `limite`, etc. En tales casos, la propiedad de conservación se cumple respecto de uno de los argumentos, de forma que los demás se mantienen "congelados", actuando como expectadores pasivos. Los argumentos (excepto uno) vienen fijados por unificación con una subexpresión del objetivo. Así, por ejemplo, el operador `limite` toma tres argumentos: una expresión de valor real dependiente de una variable real, un símbolo de variable, y un número real, de forma que dadas dos expresiones $expr_1$ y $expr_2$ de valores reales, se cumple (siempre que existan los límites de $expr_1$ y $expr_2$):

$$expr_1 < expr_2 \Longrightarrow \lim_{x \to a} expr_1 \leq \lim_{x \to a} expr_2$$

En este caso, los argumentos x y a son los mismos en los dos miembros, es decir, se está aplicando la transformación $expr \xrightarrow{\mathcal{F}_{x,a}} \lim_{x \to a} expr$. Así, por ejemplo, en la resolución del problema 8, $f_1(z)$ unifica con la subexpresión $f_1(x)$ del primer miembro del objetivo. Sobre ésta actúa el operador límite, con los argumentos x y 0. De modo que los dos primeros argumentos de `limite` quedan fijos por unificación con el objetivo, y se introducen junto con el operador en los dos miembros de la hipótesis. Más que introducir `limite`, se puede decir que se introduce $\lim_{x \to 0}$.

Como se ha indicado más arriba, la propiedad de preservación de una determinada metafunción respecto de una determinada relación se enuncia en forma de implicación, pero es fundamental sin embargo que no figure como tal en la base de reglas, ya que tendría una aplicación indiscriminada. Esto se evita de la manera siguiente. PROGENES permite que el usuario enuncie la propiedad en forma de implicación en la base de conocimiento. En el momento de cargar la base, el sistema la detecta y la reconvierte en un hecho con una sintaxis específica, como veremos a continuación.

Pero antes observemos que es posible que las propiedades de conservación de un operador sean ciertas sólo bajo determinadas condiciones. Por ejemplo, la imagen mediante una función monótona mantiene o invierte las desigualdades según la función sea creciente o decreciente. Es decir,

$$\begin{aligned} f \textit{ creciente} &\Rightarrow (\forall x, y,\ x < y \Rightarrow f(x) < f(y)) \\ f \textit{ decreciente} &\Rightarrow (\forall x, y,\ x < y \Rightarrow f(x) > f(y)) \end{aligned}$$

La preservación o inversión de las desigualdades mediante el producto por un número depende asimismo del signo del número. Por lo tanto, en la práctica, este tipo de propiedades se enuncian en la forma:

$$\begin{aligned} &\forall a_1, \ldots, a_n,\ \psi(a_1, \ldots, a_n) \Rightarrow \\ &(\forall x, y,\ x \mathcal{R}_1 y \Rightarrow \mathcal{F}(a_1, \ldots, a_{j-1}, x, a_j \ldots, a_n)\ \mathcal{R}_2\ \mathcal{F}(a_1, \ldots, a_{j-1}, y, a_j, \ldots, a_n)) \end{aligned}$$

donde $a_1, \ldots, a_n$ juegan un papel "pasivo" en esta propiedad, y ψ indica una condición sobre estos argumentos pasivos.

Cuando el sistema encuentra una expresión del tipo de la anterior, ya sea directamente en la base de conocimiento, o bien como consecuencia de alguna manipulación deductiva, antes de hacer nada con ella, la reconvierte en un hecho con una estructura predefinida, por medio del metametapredicado preserva, de uso interno del sistema. Este predicado no tiene definición procedural, y su utilidad se restringe a esta heurística. Esta reformulación de las expresiones es previa a todos los mecanismos de tratamiento de fórmulas, y permite que fórmulas como la anterior se apliquen exclusivamente por medio de la metaregla (XI). Una propiedad como la enunciada anteriormente se reformula de la siguiente forma:

preserva $(\mathcal{F},\ \mathcal{R}_1,\ j,\ \mathcal{R}_2,\ \lambda(a_1, \cdots, a_n).\psi(a_1, \cdots, a_n))$

donde $\mathcal{R}_1$ es la relación original, $\mathcal{R}_2$ es la resultante, j es el lugar que ocupa el argumento sobre el cual se cumple la propiedad de conservación, y en último lugar se indica un predicado en forma de λ-expresión, que se debe aplicar a los argumentos pasivos que aparecen en el caso particular en que se vaya a aplicar la heurística, para comprobar si se cumplen las condiciones necesarias para que se de la propiedad expresada.

Así por ejemplo, a partir de la información contenida en la base de conocimiento sobre aritmética, se derivan hechos como los siguientes:

preserva ($\times$, $<$, 1, $<$, $\lambda a.(a > 0)$)
preserva ($\times$, $<$, 2, $<$, $\lambda a.(a > 0)$)
preserva ($\times$, $<$, 1, $>$, $\lambda a.(a < 0)$)
preserva ($\times$, $<$, 2, $>$, $\lambda a.(a < 0)$)
...

que indican que *'multiplicar por un número positivo'* es una operación que no altera la relación $<$, mientras que *'multiplicar por un número negativo'* la invierte.

La base relativa a funciones reales da lugar a hechos similares, como:

preserva (imagen, $<$, 2, $>$, $\lambda a.$estrict_decr(a))
preserva (limite, $<$, 1, $\leq$, True)
preserva (limite_inferior, $<$, 1, $\leq$, True)
preserva (limite_superior, $<$, 1, $\leq$, True)
preserva (integral, $<$, 1, $\leq$, True)
...

Este tipo de hechos indican así qué operadores preservan qué relaciones de qué manera respecto de cuál argumento y bajo qué circunstancias, y se utiizan para determinar si se puede o no introducir un determinado operador en los dos argumentos de una relación binaria. Así pues, el sistema utiliza en la práctica una versión de la metaregla (XI) donde, en base a estos hechos, se comprueba que los argumentos fijos del operador que se quiere introducir verifican las condiciones que acompañan a la propiedad.

4.3.5 Substitución

Una regla de inferencia comúnmente utilizada en la demostración automática de teoremas es:

$$\frac{\Gamma, \psi, \psi_{[y/x]}, x = y \vdash \varphi}{\Gamma, \psi, x = y \vdash \varphi}$$

Esta regla de inferencia está basada en el teorema de substitutividad de la igualdad para predicados:

$$\forall x, y,\ x = y \Longrightarrow (\varphi_{[x/s]} \Leftrightarrow \varphi_{[y/s]}) \qquad \text{(XII)}$$

donde x, y, y s son términos, y φ es una fórmula.

Sin embargo, esta regla por sí sola no permite resolver un problema tan común como $x = y \ \vdash \ f(x) = f(y)$. Las heurísticas de introducción de operadores descritas más atrás proporcionan a PROGENES un medio eficaz para demostrar este tipo de resultados.

Por otra parte, un problema como $\mathcal{P}(x), x = y \ \vdash \ \mathcal{P}(y)$, no se puede resolver con las heurísticas de introducción de operadores, si $\mathcal{P}$ no es una igualdad y x e y no son canónicas.[18] De modo que necesitamos un mecanismo de substitución como la regla anterior o similar, que complemente la acción de las heurísticas de introducción de operadores.

Igual que la metaregla (VIII) se construye a partir de la adición de condiciones de control de un teorema de la lógica, PROGENES incluye una metaregla de substitución semejante a la presentada al principio de este apartado, pero con restricciones adicionales. En este caso hay una diferencia respecto a la metaregla (VIII), en cuanto a que el teorema (XII), sobre el que se substenta la regla que vamos a presentar aquí, no supone un peligro de recursión infinita si φ no es una igualdad, ya que los hechos inferidos aplicando el teorema no puede instanciar la parte izquierda de la implicación,

[18] Si por ejemplo y fuese canónica, según hemos visto en el apartado 4.3.1, el sistema crearía la regla de reescritura $x := y$, de manera que la hipótesis se convertiría en $\mathcal{P}(y)$, y el objetivo quedaría demostrado automáticamente. Si fuese canónica x, se crearía la reescritura inversa, y el objetivo se convertiría en $\mathcal{P}(x)$, que está en las hipótesis.

al contrario de lo que ocurre con el teorema (VII). Sin embargo, su aplicación exhaustiva puede suponer una duplicación excesiva de los hechos si la base de hechos contiene un gran número de igualdades, dando lugar a una expansión posiblemente innecesaria del espacio de búsqueda. Es por ello que incluimos en esta metaregla metacondiciones similares a las de (VIII), que exigen que el principio subyacente sólo se aplique cuando haya indicios que aconsejen:

$$\frac{\Gamma, x = y, \psi, \psi_{[y/s],\sigma} \vdash \varphi}{\Gamma, x = y, \psi \vdash \varphi} \quad \boxed{(s,\sigma) \leftarrow aparece\,(x,\varphi)} \qquad \text{(XIII)}$$

Esta metaregla tiene no obstante una prioridad baja. De hecho, gran parte de los mecanismos e incluso de la filosofía de PROGENES están destinados a evitar la utilización de esta regla. Siempre es preferible, si es posible, utilizar mecanismos más directos para substituir igualdades, ya sea por medio de reglas de reescritura que el propio sistema define a partir de ciertas igualdades (apartado 4.3.1), de la codificación por el usuario de definiciones y propiedades de conceptos en forma de procedimientos asociados a las metafunciones, o del tratamiento de la igualdad mediante rutinas de resolución de ecuaciones. Se trata de hacer aquellas substituciones que realmente interesan (selección y direccionamiento), de la manera más rápida posible (proceduralidad), y de forma irreversible (reescritura).

4.4 Heurísticas para tipos específicos

PROGENES incluye también heurísticas y metareglas que hacen referencia a objetos de tipo concreto, como funciones o ecuaciones, y están por tanto relacionadas con un capítulo específico del dominio, o con una clase de problemas.

4.4.1 Igualdad entre funciones

En Matemáticas se puede definir la igualdad entre dos funciones f y g como:[19]

$$f = g \iff \mathcal{D}_f = \mathcal{D}_g \ \wedge\ \mathcal{R}_f = \mathcal{R}_g \ \wedge\ \forall x \in \mathcal{D}_f, f(x) = g(x)$$

[19]Recordemos que $\mathcal{D}_f$ significa el dominio de f, y $\mathcal{R}_f$ su rango.

Esta definición da lugar a un paralelismo entre las propiedades sobre funciones y las propiedades sobre sus imágenes. Así, las igualdades del tipo *"$f(x) = \cdots$ para todo x del dominio de f"* son a menudo utilizadas por los matemáticos como definición de una función, esto es, como si la igualdad fuese *"$f = \cdots$"*. Aquí hay un abuso de lenguaje, muy habitual en Matemáticas, que consiste en considerar f y $f(x)$ indistintamente, como si fueran la misma cosa, de tal forma que las igualdades en términos de imágenes pueden ser interpretadas como igualdades entre funciones y viceversa. Esto se ha visto en varios de los ejemplos mostrados. La definición de la igualdad entre funciones a partir de la igualdad entre números reales establece un paralelismo entre ambos tipos de igualdades que garantiza que el abuso de lenguaje no crea ningún conflicto. Sin embargo, para un sistema como PROGENES, la diferencia entre una función y un número es substancial, y por ello es necesario un mecanismo explícito que nos permita pasar de una cosa a la otra.

Como vamos a ver a continuación, éste es un ejemplo de una definición de concepto que no se codifica como metafunción, ya que nos puede interesar disponer de las dos igualdades (equivalentes) en un mismo problema, como hemos visto ya en varios ejemplos. Unas veces aplicamos la deducción en una dirección, y otras veces en la otra. En esta ocasión, el conocimiento no se debe direccionar. Veamos pues de qué manera se aplica esta definición.

En las hipótesis

Cuando una hipótesis es una igualdad entre objetos de tipo `funcion`, PROGENES añade otra igualdad en términos de las imágenes de las funciones. He aquí la metaregla que lo realiza:

$$\frac{\Gamma, u = v, \mathcal{D}_u = \mathcal{D}_v, \mathcal{R}_u = \mathcal{R}_v, \forall x \in \mathcal{D}_u\ u(x) = v(x) \vdash \varphi}{\Gamma, u : funcion = v : funcion \vdash \varphi}$$

En esta regla u y v pueden ser cualquier expresión de tipo `funcion`, es decir, en particular puede ser una suma, producto, composición, etc., de funciones.

Inversamente, si en un problema aparece un hecho como $\forall x f(x) = \operatorname{sin} x$, se crea la igualdad equivalente $f = \texttt{funcion\_real}(x, \operatorname{sin} x)$. La metaregla que realiza esta operación es la siguiente:

$$\frac{\Gamma, \forall x \in \mathcal{D}_f \ f(x) = expr, f = funcion(\mathcal{D}_f, \mathcal{R}_f, \lambda x.expr) \ \vdash \varphi}{\Gamma, \forall x \in \mathcal{D}_f \ f(x) = expr \ \vdash \varphi}$$

Dado que esta regla se utiliza con mucha frecuencia para el caso particular en que f es una función real (lo hemos visto en los ejemplos), el sistema incluye también una versión especializada para este caso, evitando así tener que simplificar `funcion(`$\mathbb{R}$, $\mathbb{R}$, $\lambda x.expr$`)` a `funcion_real(`x`,`$expr$`)`.

$$\frac{\Gamma, \forall x \in \mathcal{D}_f \ f(x) = expr, f = funcion\_real\,(x, expr) \ \vdash \varphi}{\Gamma, \forall x \in \mathcal{D}_{f:funcion\_real} \ f(x) = expr \ \vdash \varphi}$$

Para evitar redundancias, se añade a la regla de arriba la condición de que f no sea una función real.

Estas metareglas han sido extensamente utilizadas por PROGENES en la resolución de problemas relativos a funciones, y su utilidad queda ilustrada en los diversos problemas que se muestran en este trabajo.

Buscar una función

Hemos visto cómo trata PROGENES el caso en que aparecen igualdades que involucran funciones en los hechos. Sin embargo esta cuestión surge igualmente cuando se busca una función, o cuando el objetivo es una igualdad entre funciones. Si por ejemplo buscamos una función real f, nos damos por satisfechos si conseguimos determinar el valor de $f(x)$ para todo x (es decir, para un x genérico). De hecho es una de las maneras habituales de determinar una función (otra puede ser por ejemplo resolver una ecuación diferencial). Y si tenemos que demostrar una igualdad entre funciones, es muy posible que lo intentemos en términos de las imágenes, aunque primero podemos intentar obtener directamente la igualdad funcional. PROGENES incluye sendas metareglas para estos casos:

$$\Gamma \vdash t : funcion = u : funcion \ \rightsquigarrow \ \Gamma \vdash \begin{cases} \mathcal{D}_t = \mathcal{D}_u \\ \wedge \ \mathcal{R}_t = \mathcal{R}_u \\ \wedge \ \forall x \in \mathcal{D}_t, t(x) = u(x) \end{cases} \qquad \text{(XIV)}$$

$$\frac{\Gamma, x_1 \in \mathcal{D}_t \overset{\Sigma, t, t(x_1)}{\vdash} \varphi}{\Gamma \overset{\Sigma, t:funcion}{\vdash} \varphi} \qquad \text{(XV)}$$

Cuando el objetivo de un problema es demostrar una igualdad entre dos expresiones funcionales, la primera metaregla cambia el objetivo aplicando la definición de igualdad entre funciones. Cuando tenemos que hallar un objeto de tipo funcion, la segunda metaregla crea una constante que denota un objeto perteneciente al dominio de la expresión de tipo función, de forma que añade a la lista de objetos buscados la imagen de la función sobre este elemento genérico. Al modificar el objetivo, la primera regla altera de forma significativa el problema, mientras que la segunda sólo añade algo, sin eliminar nada de lo que había. Este tipo de razonamiento se utiliza en la resolución del problema 7.

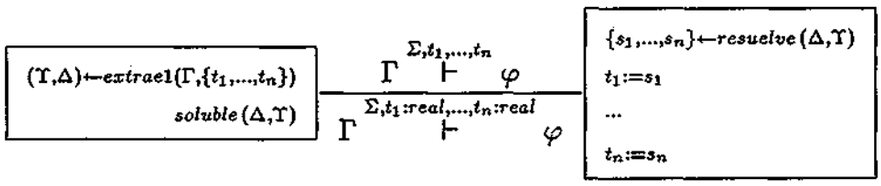
4.4.2 Resolución de ecuaciones

Muchos problemas de Cálculo involucran funciones y sus derivadas. Asimismo, muchos problemas de Mecánica o de Geometría Analítica se pueden reescribir en términos de números, funciones reales y ecuaciones algebraicas o diferenciales (ver [CAS92]), expandiendo si es necesario determinados objetos en términos de sus componentes. Como consecuencia de todo esto, muchos problemas se pueden reducir a un sistema de ecuaciones numéricas o funcionales (diferenciales) que se pueden resolver por técnicas de manipulación algebraica.

PROGENES dispone de un módulo de cálculo formal capaz de resolver una amplia variedad de problemas de este tipo. Sin embargo, el sistema debe primero darse cuenta de que un determinado problema se puede tratar con estas técnicas. Los problemas que se pueden resolver por medio de ecuaciones son de dos tipos: problemas en que se buscan objetos de tipo `real` o `funcion`, y problemas en los que hay que demostrar una igualdad entre objetos de dichos tipos.

Objetos buscados

En el primer caso, se utiliza la siguiente regla:



Esta metaregla mira cuáles de los objetos buscados son números reales, y busca en los hechos un sistema de ecuaciones que permita resolverlos. Para ello utiliza una heurística de extracción de ecuaciones para incógnitas, llevado a cabo por la función del sistema *extrael*, que consiste en lo siguiente. En primer lugar, se seleccionan aquellas igualdades de la base de hechos en las que aparece alguna de las incógnitas $t_1, \ldots, t_n$ como término *soluble* (decimos que un término es soluble en una ecuación si se puede despejar por métodos algebraicos). Después se aplica de nuevo el mecanismo a los demás términos u_j solubles que aparezcan en estas ecuaciones y *que contengan alguna incógnita*, y así sucesivamente, hasta obtener un sistema de ecuaciones $\Delta = \{\psi_1, \ldots, \psi_m\}$ respecto de los términos $\Upsilon = \{u_1, \ldots, u_k, t_1, \ldots, t_n\}$ (donde se incluyen las incógnitas originales t_j más los términos u_j que se han ido encontrando en las ecuaciones). Una vez seleccionado el sistema, se comprueba que sea soluble. Si no se hubiesen encontrado ecuaciones que involucren a los objetos buscados, la función *extrael* devuelve `False`, y la regla no se aplica.

Recuérdese que en las explicaciones de este capítulo no diferenciamos entre hechos y condiciones. Tal y como se indicó al principio de este capítulo, cuando hablamos de hechos, incluimos también los elementos del item **CONDICIONES** correspondiente al caso en que se hace razonamiento basado en el objetivo, de forma que la extracción de ecuaciones busca también en las condiciones del problema. De hecho, teniendo en cuenta que estamos hablando de problemas de tipo *hallar*, lo más probable es que todas las ecuaciones provengan de este item.

Una vez seleccionadas las ecuaciones Δ, se resuelven respecto de los términos Υ. Las soluciones dan lugar a las reglas de reescritura correspondientes, que se aplican al problema entero. Las ecuaciones utilizadas se eliminan de la base de hechos, ya que son equivalentes a las soluciones obtenidas (de hecho, al aplicar la substitución evalúan a `True`).

Además de substituir las soluciones en todo el problema, los valores obtenidos para los objetos buscados se guardan en el item **SOLUCIONES**, por si los pide explícitamente el problema, o hay que transmitirlos a otro subproblema.

Obsérvese que el sistema de ecuaciones no tiene por que resolver todas las incógnitas: puede haber más incógnitas que ecuaciones. En tal caso, el mecanismo permite al menos eliminar una parte de las incógnitas (al obtener soluciones en términos de las incógnitas restantes), de forma que cuando se hallen las incógnitas que quedan, se habrán resuelto también las eliminadas.

Los términos t_j y u_j que se toman como incógnitas del sistema de ecuaciones no son necesariamente variables, es decir, símbolos atómicos, sino que pueden ser expresiones arbitrarias, con la única condición de que sean de tipo número. Aunque resolver ecuaciones en términos de expresiones complejas puede resultar un tanto incómodo para una persona, fundamentalmente por lo farragoso de la notación, no representa ninguna dificultad particular para un ordenador, ni por tanto para PROGENES. Así como las personas tendemos a introducir nuevas variables que representan a estas expresiones, el sistema se maneja directamente con las expresiones completas, por complicadas que sean.

Los problemas en que se buscan funciones pueden resolverse igualmente mediante un sistema de ecuaciones funcionales, de modo que el sistema incluye otra metaregla análogas a la anterior para el caso $t_j : funcion\_real$. Si las ecuaciones son diferenciales, se invocará a una rutina específica para este tipo de ecuaciones.

Demostrar una igualdad

Como ya se comentado, los problemas en que se quiere demostrar una igualdad tienen mucho en común con aquéllos en los que se busca un objeto, y en particular se pueden resolver mediante la manipulación de ecuaciones. PROGENES incluye una metaregla que permite utilizar resolución de ecuaciones para demostrar una igualdad:

$$\frac{(\Upsilon,\Delta)\leftarrow extrae2(\Gamma,\{t,u\}) \quad soluble(\Delta,\Upsilon) \qquad \Gamma \vdash t = u}{\Gamma \vdash t : real = u : real} \qquad \begin{array}{l}\{s_1,s_2\}\leftarrow resuelve(\Delta,\Upsilon)\\ t{:=}s_1\\ u{:=}s_2\end{array}$$

Aquí el algoritmo de extracción de ecuaciones es distinto al que corresponde al caso anterior (objetos buscados). La función *extrae2* empieza buscando todas las ecuaciones que involucran a t y u, y hace lo mismo para cada término soluble no canónico que aparezca en estas ecuaciones, y así sucesivamente. En este caso, se busca un sistema respecto de *todas* las expresiones no canónicas recursivamente involucradas, no sólo

respecto de aquéllas que contienen incógnitas. El motivo es que si pretendemos hallar objetos, cualquier solución en la que no aparezcan incógnitas es válida, mientras que en este caso, no tiene por qué ser así. Considérese, por ejemplo, el caso en que queremos demostrar $x_1 = y_1$, sabiendo que $x_1 = y_1 + k_1$, y $k_1 = -k_1$; si no consideramos k_1 también como variable de la ecuación, nos daríamos por satisfechos con la solución $x_1 \rightarrow y_1 + k_1$, que evidentemente no resuelve el problema. Es necesario pues considerar k_1 como variable del sistema, de manera que se selecciona también la ecuación $k_1 = -k_1$, y se resuelve el sistema formado por las dos ecuaciones, obteniendo $k_1 := 0$, $x_1 := y_1$, de forma que el objetivo resulta `True`. Esta regla se ha utilizado por ejemplo en el problema 4 mostrado en el capítulo 3.

La metaregla se podría generalizar de forma que el sistema de ecuaciones se plantease no sólo respecto de los dos miembros del objetivo, sino respecto de términos solubles de éste, o incluso cualquier subexpresión. No hemos encontrado hasta el momento en la práctica problemas que justifiquen el gasto adicional que supone la inclusión de esta posibilidad, por lo que sólo utilizamos la versión más restrictiva, presentada anteriormente. Se podría considerar asimismo una metaregla que extendiese el mecanismo para los elementos del item **CLASE OBJETIVO**, es decir, que resolviese ecuaciones respecto de un elemento de este item, pero tampoco hemos visto hasta el momento que ésto sea necesario.

Ejemplo: resolución del problema 7

A modo de ejemplo de estas heurísticas, veamos la resolución del

> **Problema 7.** *Demostrar que cualquier función de* $\mathbb{R}$ *en* $\mathbb{R}$ *se puede escribir como la suma de una función par y otra impar.*

El enunciado da lugar al siguiente estado de la base de datos:

OBJETIVO : $\forall f : \mathbb{R} \rightarrow \mathbb{R}$, $\exists g : funcion\_real, h : funcion\_real$,
tales que g *par* $\wedge$ h *impar* $\wedge$ $f = g + h$

Los cuantificadores universal y existencial se eliminan del objetivo, creando la constante f_1 y las incógnitas g y h, todas ellas de tipo `funcion_real`.

OBJETIVO : $g\ par \wedge h\ impar \wedge f_1 = g + h$

INCOGNITAS :

$g : funcion\_real$,
$h : funcion\_real$

CONSTANTES : $f_1 : funcion\_real$

De acuerdo con la heurística del razonamiento basado en el objetivo (el problema es del tipo *demostrar* $\exists x P$), el sistema supone que son ciertos los hechos a demostrar sobre g y h:

CONDICIONES :

(1) $g\ par$
(2) $h\ impar$
(3) $f_1 = g + h$

La igualdad (3) proporciona una ecuación que involucra a las incógnitas, y que se puede resolver respecto de cualquiera de ellas, de modo que la heurística descrita en este apartado se aplica aquí, resolviendo (3) respecto de g (resolviendo respecto de h se obtendría una demostración similar del problema). Utilizando la solución resultante, $g := f_1 - h$, se elimina g en el problema, substituyéndola por la solución (el hecho (3) desaparece):

INCOGNITAS : $h : funcion\_real$

SOLUCIONES : $g \rightarrow f_1 - h$

CONDICIONES :

(1') $f_1 - h\ par$
(2) $h\ impar$

La metaregla (XV) descrita en el apartado 4.4.1 actúa aquí, creando la constante $x_1 \in \mathbb{R}$, y añadiendo $h(x_1)$ a la lista de objetos buscados. La idea es que es lo mismo hallar h que hallar su imagen sobre un término genérico x_1.

INCOGNITAS :

$h : funcion\_real$
$h(x_1)$

CONSTANTES :

...

$x_1 \in \mathbb{R}$

Al no haber más reglas aplicables, se activa y evalúa la definición procedural de las metafunciones par, impar e imagen de resta de funciones, obteniendo:

CONDICIONES :

$(1'')\ \forall x \in \mathbb{R},\ f_1(x) - h(x) = f_1(-x) - h(-x)$
$(2')\ \forall x \in \mathbb{R},\ h(x) = -h(-x)$

Los hechos universales $(1'')$ y $(2')$ se instancian con la constante x_1, con lo que se obtiene:

CONDICIONES :

...

$(3)\ f_1(x_1) - h(x_1) = f_1(-x_1) - h(-x_1)$
$(4)\ h(x_1) = -h(-x_1)$

Ahora tenemos que $h(x_1)$, que es un objeto buscado, aparece en un sistema de ecuaciones, el formado por (3) y (4), que puede resolver respecto de las dos expresiones que contienen incógnitas, $h(x_1)$ y $h(-x_1)$. Aplicando nuevamente la heurística descrita en este apartado, esta vez para objetos de tipo `real`, se resuelve el sistema, y se substituyen las soluciones en todo el problema:

SOLUCIONES :

$h(x) \rightarrow \frac{1}{2}(f(x) - f(-x))$
$h \rightarrow funcion\_real\,(x, \frac{1}{2}(f(x) - f(-x)))$
$g \rightarrow f_1 - h \rightarrow funcion\_real\,(x, \frac{1}{2}(f(x) + f(-x)))$

Al haberse así obtenido valores admisibles para las incógnitas del problema, el razonamiento basado en el objetivo queda prácticamente concluido; tan sólo resta comprobar que los valores cumplen las condiciones originales, lo cual resulta inmediato,

ya que por efecto de las correspondientes reglas de reescritura, éstas evalúan a True.

□

4.4.3 Heurísticas para clases de problemas

Además de las heurísticas que hemos descrito a lo largo de este capítulo, PROGENES incluye otras mucho más específicas aún, que le permiten resolver situaciones concretas.

Límites

Por ejemplo, demostrar la existencia de un límite significa en rigor demostrar que existe un número l que verifica la definición. Sin embargo, demostrar que $\forall \varepsilon > 0 \ldots$ puede ser muy difícil si ni siquiera conocemos el valor de l. Hay formas más viables de hacerlo, como por ejemplo, buscar una función que sepamos que es igual a la que tenemos, y que tenga límite en el punto. De esta forma, sabemos que si la segunda función tiene límite, la primera también lo tiene, y además el límite de la primera es igual al de la segunda. Esto se refleja en la siguiente metaregla:

$$\Gamma \vdash \mathit{existe} \lim_{x \to a} \mathit{expr} \quad \rightsquigarrow \quad \Gamma \overset{g:\mathit{funcion\_real}}{\vdash} \mathit{funcion\_real}(x, \mathit{expr}) = g \wedge \mathit{existe} \lim_{x \to a} g(x)$$

donde g es una función distinta de la inicial, es decir, se debe encontrar (u obtener) una igualdad de la base de hechos que establezca explícitamente la igualdad entre ambas. Esta regla produce asimismo como efecto lateral la regla de reescritura $\lim_{x \to a} \mathit{expr} := \lim_{x \to a} g(x)$, de forma que si el límite de f era un objeto buscado, al aplicar la regla se buscará el de g, que una vez determinado nos proporciona el valor inicialmente requerido.

Otra estrategia muy habitual para demostrar que existe el límite de una función en un punto es suponer que existe tal límite, y determinar su valor utilizando procedimientos de cálculo. Si conseguimos hallarlo, sabemos que, de existir, el límite sólo puede ser el valor obtenido, con lo cual el problema se reduce a demostrar que efectivamente el valor es el límite de la función, usando la definición en términos de ε y δ. Esta es la manera en que lo hace un matemático, y es también como lo hace PROGENES. La

heurística se lleva a cabo automáticamente y de forma implícita: al evaluar el objetivo, el sistema aplica las reglas de reescritura asociadas al concepto de `limite`, de forma que es posible que resulte un número, o el límite de otra función que conocemos o sabemos que existe. En la práctica, las rutinas de cálculo que utiliza PROGENES están construidas de tal manera que, de llegar a un resultado, garantizan que efectivamente es el límite buscado, de forma que no es necesario demostrar que el valor hallado es el límite de la función.

Desigualdades

PROGENES incluye igualmente otras metareglas que aplican métodos específicos para casos concretos. En el apartado 4.3.4 hemos visto una heurística general que resulta muy útil en particular para demostrar desigualdades. El sistema incluye también otras heurísticas más específicas de demostración de desigualdades. Una de ellas consiste en que si la desigualdad contiene una incógnita que proviene de la eliminación de un cuantificador existencial en el objetivo, entonces se utiliza un módulo de resolución de inecuaciones que devuelve el conjunto de valores para los que se cumple el objetivo.

Por otra parte, si en la desigualdad no aparecen incógnitas, y aparece una constante x_1 de tipo `real` creada por el sistema (es decir, resultante de la eliminación de un $\forall$), PROGENES busca en la base de datos hechos que indiquen restricciones sobre x_1 del tipo de pertenencia a intervalos (o lo que es lo mismo, desigualdades). En caso de encontrarlos, llamemos A a la intersección de los intervalos (caso de existir tales restricciones). Dada esta situación, estamos ante un problema del tipo de demostrar $\forall x \in A,\ expr_1 < expr_2$, y por tanto, una metaregla del sistema define un problema de minimización de $expr_1 - expr_2$ en un conjunto A, comprobando que el mínimo absoluto, de existir, es menor que 0, en cuyo caso el problema queda resuelto. Para determinar el mínimo de la expresión en x_1, el sistema utiliza los métodos usuales, buscando los puntos críticos de la función, etc., métodos que forman parte de la definición procedural de la metafunción `minimo`. Si la minimización falla (bien por que no existe un mínimo absoluto, o porque no es capaz de determinarlo el sistema, o incluso porque el valor hallado no cumplé la desigualdad), se vuelve a la situación inicial, intentando demostrar el objetivo por otros medios.

Además de estas heurísticas para desigualdades, el sistema incluye otra, de menor prioridad, que permite aplicar en backward chaining y de forma controlada la transitividad de las relaciones de orden.

$$\Gamma, x < y \vdash x < u \;\rightsquigarrow\; \Gamma, x < y \vdash y \leq u$$

Otras metareglas análogas permiten un tratamiento similar de las demás relaciones de desigualdad. Obsérvese que si esta propiedad se incluyese en forma de regla de producción, daría lugar a una generación expansiva de subobjetivos por backward chaining, ya que cada vez que tengamos que demostrar una desigualdad se aplicaría la regla, dándose además el caso de que se puede hacer nuevamente backward con la misma regla sobre los subobjetivos generados por ella misma, entrando en una expansión infinita.

Por otra parte, la aplicación de la transitividad en forward la realiza automáticamente y de forma implícita el motor de inferencia, que almacena las desigualdades formando cadenas ordenadas, que el algoritmo de unificación maneja convenientemente cada vez que tiene que comprobar una desigualdad determinada.

4.5 Ejemplos

A modo de ilustración de las técnicas descritas en este capítulo, se describe a continuación la resolución por PROGENES de los problemas listados en el capítulo 1 que no han sido mostrados todavía.

4.5.1 Problema 2

Problema 2. *Dadas las funciones* $f : A \rightarrow B$, $g, h : C \rightarrow A$, *demostrar que* $f \circ g = I$, $h \circ f = I \Rightarrow h = g$.

La idea de la resolución de este ejemplo consiste en componer $h \circ f \circ g$, de forma que por la propiedad asociativa se obtiene $I \circ g = h \circ I$, de donde se demuestra el objetivo. En este problema hay un aspecto delicado relativo a los dominios y rangos de las funciones, que hay que tratar cuidadosamente. Veámoslo paso a paso.

OBJETIVO : $h_1 = g_1$

HECHOS :

(1) $f_1 : A_1 \rightarrow B_1$
(2) $g_1, h_1 : C_1 \rightarrow A_1$
(3) $f_1 \circ g_1 = I$
(4) $h_1 \circ f_1 = I$

CONSTANTES :

f_1 : *funcion*
g_1 : *funcion*
h_1 : *funcion*
A_1 : *conjunto*
B_1 : *conjunto*
C_1 : *conjunto*

La representación interna de los hechos (1) y (2) consiste en las correspondientes igualdades en términos de dominio y rango de f_1, g_1 y h_1.

Las igualdades entre funciones (3) y (4) dan lugar a las igualdades correspondientes entre sus imágenes (obsérvese que $\mathcal{D}_{f_1 \circ g_1}$ y $\mathcal{D}_{h_1 \circ f_1}$ se simplifican (se reescriben automáticamente) a $\mathcal{D}_{g_1}$ y $\mathcal{D}_{f_1}$):

HECHOS :

(5) $\forall x \in \mathcal{D}_{g_1},\ f_1(g_1(x)) = x$
(6) $\forall x \in \mathcal{D}_{f_1},\ h_1(f_1(x)) = x$

La regla añade también las correspondientes igualdades entre los dominios y los rangos, que no hacen falta en este problema.

Ahora el sistema aplica la metaregla (XIV) del apartado 4.4.1 para demostrar igualdades entre funciones, reduciendo el problema a:

OBJETIVO : $\mathcal{D}_{h_1} = \mathcal{D}_{g_1} \ \wedge\ \mathcal{R}_{h_1} = \mathcal{R}_{g_1} \ \wedge\ \forall x \in \mathcal{D}_{h_1}\ h_1(x) = g_1(x)$

La conjunción se descompone en tres subproblemas, de los cuales los dos primeros son consecuencia directa de (2). Después de eliminar la cuantificación, el tercero es:

OBJETIVO : $h_1(x_1) = g_1(x_1)$

HECHOS :

...

(7) $x_1 \in \mathcal{D}_{h_1}$

CONSTANTES :

...

x_1 : *objeto*

El problema se resuelve por introducción de operadores en (5) y (6) por distintas unificaciones con los miembros del objetivo. Pero para poder unificar las expresiones, tienen que cumplirse las condiciones de pertenencia al conjunto que aparece en las cuantificaciones universales. Para ello hay que utilizar las condiciones propias del concepto de `funcion`, junto con el hecho de que varios de los conjuntos que aparecen en este problema son iguales.

Las igualdades correspondientes a (1) y (2) se proceduralizan, de acuerdo con la heurística descrita en el apartado 4.3.1, al ser uno de sus miembros una llamada a la función de acceso `dominio`, y el otro una expresión sin incógnitas, de forma que se crean las reglas de reescritura:

$$\begin{array}{lll} \mathcal{D}_{f_1} := A_1 & \mathcal{D}_{g_1} := C_1 & \mathcal{D}_{h_1} := C_1 \\ \mathcal{R}_{f_1} := B_1 & \mathcal{R}_{g_1} := A_1 & \mathcal{R}_{h_1} := A_1 \end{array}$$

cuya aplicación a los hechos (5), (6) y (7) da lugar respectivamente a:

(5') $\forall x \in C_1,\ f_1(g_1(x)) = x$
(6') $\forall x \in A_1,\ h_1(f_1(x)) = x$
(7') $x_1 \in C_1$

Por otra parte, tal y como se indicó en el capítulo anterior, cada vez que en un problema aparece una constante f_1 de tipo `funcion`, se añaden a los hechos las condiciones que corresponden a este tipo de objeto, entre las cuales figura el hecho universal $\forall x \in \mathcal{D}_{f_1}\ f_1(x) \in \mathcal{R}_{f_1}$. En este caso, tenemos tres funciones, luego tendremos otros tantos hechos como éste, de forma que después de aplicar las reglas de reescritura indicadas anteriormente, tenemos:[20]

[20]En otros problemas aquí mostrados se omite este aspecto cuando no tiene relevancia para la resolución del problema.

(8) $\forall x \in A_1,\ f_1(x) \in B_1$
(9) $\forall x \in C_1,\ g_1(x) \in A_1$
(10) $\forall x \in C_1,\ h_1(x) \in A_1$

Estos hechos universales se instancian con las constantes del problema que cumplen la restricción que se indica en la cuantificación. En este caso, la única posibilidad de instanciación viene dada por (7'), de forma que (9) y (10) se instancian con $[x_1/x]$:

(11) $g_1(x_1) \in A_1$
(12) $h_1(x_1) \in A_1$

Si la condición asociada al concepto de función se hubiese representado como regla $(x \in \mathcal{D}_{f_1} \Rightarrow f_1(x) \in \mathcal{R}_{f_1})$, se podría formar un ciclo infinito cuando $\mathcal{D}_{f_1}$ y $\mathcal{R}_{f_1}$ son iguales, o cuando, como en este caso, tenemos una serie de funciones en las que existen coincidencias entre dominios y rangos, de forma que podríamos deducir hechos de la forma $f_1(h_1(f_1(\cdots(h_1(x_1))\cdots))) \in B_1$ de manera indefinida. Al representar la condición en forma de hecho universal, limitamos su aplicación a las constantes del problema, objetos atómicos por tanto, de forma que nunca se aplica sobre expresiones compuestas como $h_1(x_1)$, con lo cual se evita el peligro de recursividad desde el primer momento.[21]

Ahora ya sí se puede aplicar la regla de introducción de operadores respecto del objetivo. Por un lado, tenemos que el segundo miembro de (6') unifica con ambos miembros del objetivo, con las substituciones $[g_1(x_1)/x]$ y $[h_1(x_1)/x]$ respectivamente, lo cual es posible gracias a los hechos (11) y (12). Aplicando la metaregla con las dos posibles instanciaciones, se obtiene:

(13) $h_1(f_1(g_1(x_1))) = g_1(x_1)$
(14) $h_1(f_1(h_1(x_1))) = h_1(x_1)$

Obsérvese que la metaregla no se podría haber aplicado sobre el hecho (5') con esa instanciación, ya que no consta que $g_1(x_1)$ ni $h_1(x_1)$ sean elementos del conjunto C_1. En cambio, la regla *sí* se puede aplicar a (5') mediante aplicación de g_1 ó h_1, con $\sigma \rightarrow [x_1/x]$, ya que (7') garantiza que x_1 cumple la restricción impuesta en la cuantificación. Los hechos resultantes son:

[21] Otra forma de evitar que la recursividad produzca un ciclo infinito es tomar nota de los objetos que van siendo creados, junto con los objetos a partir de los cuales han sido creados, de forma que el sistema pueda detectar estos bucles.

(15) $g_1(f_1(g_1(x_1))) = g_1(x_1)$
(16) $h_1(f_1(g_1(x_1))) = h_1(x_1)$

La metaregla no se puede aplicar a (6') con la misma substitución, ya que la instanciación de x por x_1 no es legal en este caso, al no cumplirse $x_1 \in A_1$.

Los hechos (13) y (15), situados por el sistema en la misma clase de equivalencia, dan la solución.

□

4.5.2 Problema 3

Problema 3. *Dadas $f : A \to B$ y $g : B \to A$, tales que $f \circ g = I$, la función identidad, demostrar que g es inyectiva y f es sobreyectiva.*

El sistema clasifica la información dada en el enunciado (igual que en problemas anteriores, (1) expresa en forma abreviada los hechos $\mathcal{D}_{f_1} = A_1, \mathcal{R}_{f_1} = B_1$, etc.).

OBJETIVO : g_1 *inyectiva* $\wedge$ f_1 *sobreyectiva*

HECHOS :

(1) $f_1 : A_1 \to B_1$
(2) $g_1 : B_1 \to A_1$
(3) $f_1 \circ g_1 = I$

CONSTANTES :

f_1 : *funcion*
g_1 : *funcion*
A_1 : *conjunto*
B_1 : *conjunto*

Las heurísticas de preprocesamiento de fórmulas son aplicadas por el motor de inferencia a las fórmulas del problema. En este caso el objetivo, que es una conjunción, se descompone en dos subproblemas, cuyos objetivos son g_1 *inyectiva* y f_1 *sobreyectiva* respectivamente. El objetivo original será satisfecho si ambos subobjetivos son demostrados.

Subproblema 3.1

PROGENES empieza por demostrar:

OBJETIVO : g_1 *inyectiva*

Tenemos un hecho, (3), que expresa una igualdad entre funciones, de modo que se añade la igualdad correspondiente en términos de los dominios y las imágenes (también se añaden las igualdades correspondientes entre dominios y rangos, pero las omitimos aquí por no ser relevantes para la resolución del problema):

(4) $\forall z \in \mathcal{D}_{f_1 \circ g_1},\ (f_1 \circ g_1)(z) = I(z)$

La definición de imagen de una composición de funciones sobre un objeto es de aplicación directa, como ya hemos visto, y por tanto se aplica en (4). Asimismo, $\mathcal{D}_{f \circ g}$ se simplifica automáticamente (por reescritura de aplicación directa) a $\mathcal{D}_g$, de modo que (4) se reescribe como:

(4') $\forall z \in \mathcal{D}_{g_1},\ f(g(z)) = z$

De acuerdo con la regla de proceduralización de igualdades, el hecho (3) debe convertirse en regla de reescritura. En este caso, no ayuda a resolver el problema, pero tampoco supone una traba, ya que actúa después de que se obtenga (4). Lo importante es que la igualdad (4') no se proceduraliza debido a la variable de cuantificación universal z, que hace que el segundo miembro no sea canónico salvo incógnitas.

Tras intentar infructuosamente derivar alguna conclusión a partir de los hechos del problema aplicando teoremas, las reglas de rescritura asociadas a la metafunción `inyectiva` son activadas y aplicadas, y como consecuencia, el objetivo se substituye por su definición:

OBJETIVO : $\forall x, y \in \mathcal{D}_{g_1},\ g_1(x) = g_1(y) \Rightarrow x = y$

Después de eliminar los conectores del objetivo, tenemos:

OBJETIVO : $x_1 = y_1$

HECHOS :

...

(5) $x_1 \in \mathcal{D}_{g_1}$

(6) $y_1 \in \mathcal{D}_{g_1}$

(7) $g_1(x_1) = g_1(y_1)$

CONSTANTES :

...

x_1 : *objeto*

y_1 : *objeto*

El segundo miembro de (4') unifica con ambos miembros del objetivo (incluyendo el chequeo de pertenencia al conjunto $\mathcal{D}_{g_1}$), de forma que se puede aplicar la metaregla (VIII) de introducción de operadores con dos instanciaciones: $[x_1/t, y_1/u, z/x, f_1(g_1(z))/y]$ (con $s \rightarrow x_1$ y $\sigma = [x_1/z]$), y $[y_1/t, x_1/u, z/x, f_1(g_1(z))/y]$ (con $s \rightarrow y_1$ y $\sigma \rightarrow [y_1/z]$). En ambos casos, la subexpresión s de t es el mismo t: el operador introducido es la identidad (esto es, ningún operador), obteniendo instantiaciones *interesantes* del hecho (4'):

(8) $f_1(g_1(x_1)) = x_1$

(9) $f_1(g_1(y_1)) = y_1$

En este punto, una persona se daría cuenta de que el problema se puede resolver aplicando estas igualdades al hecho (7), ya que ello haría aparecer x_1 e y_1, (los dos miembros del objetivo) en una igualdad. Para ello es necesario aplicar f_1 a ambos miembros de (7), y entonces podríamos aplicar (8) y (9).

PROGENES sigue un razonamiento similar. Las igualdades (8) y (9) comparten un miembro con el hecho a demostrar, y esto va a ser lo que guíe al sistema hacia la obtención de éste. El segundo miembro de (8) y el de (9) son iguales a sendos miembros del objetivo, y como vimos en el apartado 4.3.2, esto hace que los otros miembros se incluyan en el item **CLASE OBJETIVO:**

CLASE OBJETIVO : $f_1(g_1(x_1)), f_1(g_1(y_1))$

Por otra parte, el primer miembro de (7) aparece en el primer elemento de la clase del objetivo (es decir, éste contiene una subexpresión que unifica con (que de hecho es igual a) un miembro del hecho (7)). Aplicando la heurística de introducción de operadores en su variante para la clase del objetivo (en este caso $\sigma \rightarrow [\,]$), se introduce

f_1 en ambos miembros de (7):[22]

(10) $f_1(g_1(x_1)) = f_1(g_1(y_1))$

Como ya se indicó anteriormente, PROGENES almacena las igualdades formando clases de equivalencia, de tal manera que las propiedades simétrica y transitiva de la igualdad se aplican implícitamente. Así pues, (8), (9) y (10) se sitúan en una misma clase de equivalencia, en la que aparecen x_1 e y_1, y por tanto $x_1 = y_1$ queda demostrado.

Subproblema 3.2

OBJETIVO : f_1 *sobreyectiva*

Aplicando la definición de *sobreyectiva*, tenemos:

OBJETIVO : $\forall y \in \mathcal{R}_{f_1}, \exists x \in \mathcal{D}_{f_1}$ *tal que* $f_1(x) = y$

de forma que después de eliminar todos los conectores (creando la constante y_2 y la variable x), el problema se reduce a:

OBJETIVO : $f_1(x) = y_2 \ \wedge \ x \in \mathcal{D}_{f_1}$

INCOGNITAS : x : *objeto*

HECHOS :

...

(11) $y_2 \in \mathcal{R}_{f_1}$

CONSTANTES :

...

y_2 : *objeto*

Para demostrar la conjunción, el problema se descompone en dos. El sistema empieza por demostrar:

[22] El mismo razonamiento se podría haber hecho con el otro elemento del item, $f_1(g_1(y_1))$, y el segundo miembro de (7), y se obtendría exactamente el mismo resultado.

OBJETIVO : $f_1(x) = y_2$

Como ya hemos dicho antes, los hechos (1) y (2) representan una serie de igualdades, entre ellas $\mathcal{R}_{f_1} = B_1$ y $\mathcal{D}_{g_1} = B_1$. Igual que en el problema 2, estas igualdades se convierten en reglas de reescritura, ya que un miembro es una llamada a la función de acceso `dominio`, y el otro es una expresión sin incógnitas. De modo que tenemos $\mathcal{R}_{f_1} := B_1$, y $\mathcal{D}_{g_1} := B_1$ y por lo tanto, (4') y (11) se reescriben a:

(4'') $\forall z \in B_1, f_1(g_1(z)) = z$
(11') $y_2 \in B_1$

Esto permite la unificación del objetivo con el hecho (4''), ya que (11') hace que la instanciación de z por y_2 sea admisible. La substitución resultante es $[y_2/z, g_1(z)/x]$, lo cual proporciona el valor $x \rightarrow g_1(y_2)$ para la variable buscada.

Queda demostrar el subobjetivo pendiente, $x \in \mathcal{D}_{f_1}$, con el valor $g_1(y_2)$ obtenido en el subproblema anterior para la incógnita x, es decir, el sistema tiene que demostrar $g_1(y_2) \in \mathcal{D}_{f_1}$. Aplicando las reglas de reecritura resultantes de (1) y (2), el objetivo se reescribe a $g_1(y_2) \in A_1$.

Igual que en el problema precedente, la presencia de un objeto de tipo `funcion` como g_1 hace que el sistema añada automáticamente el hecho (otro hecho análogo se añadiría para f_1):

(12) $\forall x \in B_1,\ g_1(x) \in A_1$

El objetivo se deduce por unificación con este hecho universal, con $[y_2/x]$, lo cual es posible por el hecho (11').

□

4.5.3 Problema 10

Problema 10. *Demostrar que si f es continua en 0, y para todo $x \in \mathbb{R}$, $f(x+y) = f(x) + f(y)$, entonces f es continua en todo $\mathbb{R}$.*

Una clave importante para resolver este problema de manera sencilla reside en la elección de una de las formas equivalentes de dar la definición de continuidad. Escogiendo la forma $\lim_{h\to 0} f(x+h) = f(x)$, la resolución es mucho más fácil y directa. Se podría utilizar otra base de conocimiento equivalente en la que se diera la definición de otra manera, pero el problema se puede complicar mucho más, haciendo necesaria una flexibilidad muy grande para las manipulaciones algebraicas.

En la base de datos, tenemos:

OBJETIVO : f_1 *continua en* $\mathbb{R}$

HECHOS :

(1) $f_1 : \mathbb{R} \to \mathbb{R}$
(2) f_1 *continua en* 0
(3) $\forall x, y \in \mathbb{R},\ f_1(x+y) = f_1(x) + f_1(y)$

CONSTANTES : f_1 : *funcion_real*

PROGENES comienza por incluir el objeto atómico 0 en la lista de constantes, después de lo cual, al no haber reglas aplicables, se utiliza la definición de continuidad indicada anteriormente, y tras eliminar la cuantificación universal resultante en el objetivo, tenemos:[23]

OBJETIVO : $\lim_{h\to 0} f_1(x_1 + h) = f_1(x_1)$

HECHOS :

...
(2') $\lim_{h\to 0} f_1(h) = f_1(0)$
...

CONSTANTES : $\ldots, 0, x_1 \in \mathbb{R}$

La presencia del concepto de `limite` en el hecho (2') hace que el sistema añada como hecho la condición asociada al concepto:

[23] En la práctica, en un principio la variable del límite que aparece en el hecho (2') no tiene por qué ser h, sino que procede de el símbolo que se haya utilizado en la definición de límite. PROGENES substituye sistemáticamente las variables que aparecen en los límites, de manera que se utilice la misma en todo el problema, facilitando de este modo la detección de expresiones equivalentes.

(4) $existe \lim_{h \to 0} f_1(h)$

Por otro lado, al aparecer $\lim_{h \to 0} f_1(x_1 + h)$ en el objetivo, el sistema añade automáticamente como subobjetivo que el límite exista, de acuerdo con la condición establecida en la definición del concepto de `limite` en la base de conocimiento. Aplicando la heurística descrita en el apartado 4.4.3, se plantea el subproblema:

OBJETIVO : $funcion\_real\,(h, f_1(x_1 + h)) = g \ \wedge \ existe \lim_{h \to 0} g(h)$

INCOGNITAS : $g : funcion\_real$

PROGENES empieza por el primer subobjetivo. El primer miembro del hecho (3) unifica con una subexpresión del primer miembro del objetivo, mediante $[x_1/x, h/y]$, de modo que PROGENES aplica la metaregla de introducción de operadores, obteniendo:

(5) $funcion\_real\,(h, f_1(x_1 + h)) = funcion\_real\,(h, f_1(x_1) + f_1(h))$

Esta igualdad unifica con el objetivo a demostrar, de forma que se obtiene el valor $funcion\_real\,(h, f_1(x_1) + f_1(h))$ para el objeto buscado g.

Ahora el sistema demuestra el segundo subobjetivo:

OBJETIVO : $existe \lim_{h \to 0} funcion\_real\,(h, f_1(x_1) + f_1(h))(h)$

Aplicando conocimiento procedural directo de las metafunciones `imagen` y `limite` (imagen de función explícita, límite de una suma, y límite de una constante), el objetivo se reescribe a:

OBJETIVO : $existe\ f_1(x_1) + \lim_{h \to 0} f_1(h)$

Estas propiedades procedurales de la metafunción `limite` son reglas de reescritura de aplicación directa correspondientes a manipulaciones elementales de aplicación inmediata. El primer sumando, $f_1(x_1)$, se elimina, y la existencia del segundo se demuestra directamente como consecuencia del hecho (4).

Esta comprobación preliminar de la existencia del límite sobre el cual queremos demostrar un determinado hecho tiene como efecto lateral, de acuerdo con lo indicado en el apartado 4.4.3, la creación de la regla de reescritura:

$$\lim_{h\to 0} f_1(x_1 + h) := \lim_{h\to 0} g(h)$$

que tras substituir el valor hallado para g y realizar las simplificaciones pertinentes se convierte en:

$$\lim_{h\to 0} f_1(x_1 + h) := f_1(x_1) + \lim_{h\to 0} f_1(h)$$

Una vez comprobada la existencia del límite, el sistema pasa a demostrar el objetivo pendiente:

OBJETIVO : $\lim_{h\to 0} f_1(x_1 + h) = f_1(x_1)$

Aplicando la regla de reescritura obtenida anteriormente, tenemos:

OBJETIVO : $f_1(x_1) + \lim_{h\to 0} f_1(h) = f_1(x_1)$

PROGENES aplica de forma sistemática al objetivo rutinas de simplificación algebraica, de forma que en este caso se obtiene:

OBJETIVO : $\lim_{h\to 0} f_1(h) = 0$

En este punto, teniendo en cuenta el hecho (2'), sólo nos falta saber que $f_1(0) = 0$ para resolver el problema. Esto se consigue por instanciación del hecho (3) con las constantes atómicas de tipo `real` (de acuerdo con los tipos de las variables cuantificadas en (3)) que aparecen en el problema, es decir, x_1 y 0. De todos los hechos generados de esta manera, el que nos interesa es:

(6) $f_1(0 + 0) = f_1(0) + f_1(0)$

Ahora se aplica la heurística de demostración de igualdades por resolución de ecuaciones. El primer miembro del objetivo aparece como término soluble en (2'). De este hecho se extrae el término no canónico $f_1(0)$, que a su vez aparece en la ecuación (6). De modo que se resuelve el sistema formado por (2') y (6) respecto de los términos seleccionados, y se crean las reglas de reescritura correspondientes a las soluciones obtenidas:

$$f_1(0) := 0$$
$$\lim_{h \to 0} f_1(h) := 0$$

Al aplicar la segunda regla en el objetivo, se obtiene True, lo cual concluye la demostración.

□

4.5.4 Problema 12

Problema 12. *Sea* $f : [0,1] \to [0,1]$ *una función derivable en* $[0,1]$ *tal que para todo* $x \in [0,1]$, $f'(x) \neq 1$. *Demostrar que existe exactamente un* x *en* $[0,1]$ *tal que* $f(x) = x$.

Este problema se puede considerar como no trivial para una persona. También lo es para PROGENES. En consecuencia, la mayoría de las heurísticas preferentes fallan en un principio, de forma que el problema se resuelve por mecanismos de baja prioridad, como la instanciación universal.

OBJETIVO : $\exists x \in [0,1]$ *tal que* ($f_1(x) = x \ \wedge \ \forall y \in [0,1], f_1(y) = y \Rightarrow y = x$)

HECHOS :

(1) $f_1 : [0,1] \to [0,1]$
(2) f_1 *derivable en* $[0,1]$
(3) $\forall t \in [0,1]$, $f_1'(t) \neq 1$

CONSTANTES : f_1 : *funcion_real*

Una vez más, el objetivo se descompone. Como consecuencia, se crea la variable x, que pasa a formar parte de la lista de objetos buscados, y el problema se divide en tres subproblemas (ya que $x \in [0,1]$ se añade al objetivo), de forma que cada uno toma el valor para x que se obtiene de la demostración del anterior (si se obtiene tal valor).

Pero antes de descomponer el problema, el sistema añade los objetos atómicos canónicos a la lista de constantes:

CONSTANTES : $\ldots, 0, 1$

Subproblema 12.1

OBJETIVO : $f_1(x) = x$
INCOGNITAS : $x \in \mathbb{R}$

Aplicando a (2) la regla (el teorema) que dice que si una función es derivable, entonces es continua, se obtiene:

(4) f_1 *continua en* $[0,1]$

Igual que en problemas anteriores, (1) significa $\mathcal{D}_{f_1} = [0,1]$ y $\mathcal{R}_{f_1} = [0,1]$, que se convierte en conocimiento procedural al ser canónico uno de los miembros de cada igualdad (en este caso, `dominio` no es un slot de `funcion_real`). Por otra parte, el sistema añade automáticamente las condiciones asociadas al concepto de `funcion`, de forma que tenemos:

(5) $\forall x \in [0,1],\ f_1(x) \in [0,1]$

donde se han aplicado las reglas de reescritura derivadas de (1).

Al no haber reglas que se puedan aplicar, la actividad forward se detiene. El backward chaining entra entonces en funcionamiento, y encuentra que la regla que codifica el teorema de Bolzano:

$$f, g \textit{ continuas en } [a,b] \ \wedge\ c,d \in [a,b] \wedge\ f(c) \geq g(c) \ \wedge\ f(d) \leq g(d)$$
$$\Longrightarrow \exists \xi \in [a,b] \textit{ tal que } f(\xi) = g(\xi)$$

puede dar la solución. Esto es posible porque PROGENES utiliza el resolvedor de ecuaciones extendido introducido en el apartado 3.5.1 del capítulo 3. En este caso, la conclusión del teorema y el objetivo del problema unifican mediante $[f_1/f, funcion\_real(x, x)/g, \xi/x]$.[24]

Cada hipótesis del teorema es instanciada, dando lugar a un subproblema correspondiente. El primero se satisface fácilmente con $[0/a], [1/b]$: el hecho (4) garan-

[24]Obsérvese que la unificación también es posible por medio de $[funcion\_real(x,x)/f, f_1/g, \xi/x]$, pero la instanciación correspondiente de las hipótesis del teorema no llegan a poder demostrarse.

tiza la continuidad de f_1, y una heurística específica comprueba que la identidad es continua en todas partes (es un polinomio). Falta pues hallar c y d tales que $f_1(c) \geq funcion\_real(x,x)(c)$ y $f_1(d) \leq funcion\_real(x,x)(d)$. La metafunción imagen se aplica siempre cuando la función sobre la que actúa está dada explícitamente en la forma $funcion\_real(var, image)$ (o en la forma explícita correspondiente a las funciones generales). De forma que los subobjetivos se convierten en $f_1(c) \geq c$ y $f_1(d) \leq d$ respectivamente.

Al no haber reglas que aplicar, el hecho (5) se instancia con las constantes 0 y 1:

(6) $f_1(0) \in [0,1]$
(7) $f_1(1) \in [0,1]$

En este punto, la definición procedural de la metafunción $\in$ es activada, de forma que se ejecuta en (6) y (7), dando lugar a:

(6') $0 \leq f_1(0) \leq 1$
(7') $0 \leq f_1(1) \leq 1$

Estos hechos proporcionan por unificación con cada subobjetivo los valores buscados: $c \rightarrow 0$ y $d \rightarrow 1$.

Así pues la regla se dispara, añadiendo un hecho existencial cuya cuantificación es eliminada, creando la constante $\xi_1 \in [0,1]$. El objetivo original queda así demostrado, y ξ_1 es el valor de x que lo satisface.

En este subproblema, el backward chaining permite obtener resultados que no podrían ser alcanzados sólo con forward. La razón de esto es que el fuerte interés por aplicar el teorema hace que una expresión como x se vea como función de x, a la que se aplica el teorema. De otra forma, no hay razón en principio para manejar tal expresión de esa manera. El mecanismo clave aquí, como hemos visto, está en la flexibilidad del procedimiento de unificación.

Subproblema 12.2

El hecho previamente demostrado es añadido a la base de datos, y el valor ξ_1 obtenido para x se substituye en el siguiente subproblema.

OBJETIVO : $\forall y \in [0,1], f_1(y) = y \Rightarrow y = \xi_1$

SOLUCIONES : $x \rightarrow \xi_1$

HECHOS :

...

(8) $\xi_1 \in [0,1]$

(9) $f_1(\xi_1) = \xi_1$

De nuevo se eliminan los conectores lógicos del objetivo, creando una constante $y_1 \in [0,1]$, asumiendo el hecho $f_1(y_1) = y_1$, y estableciendo $y_1 = \xi_1$ como el nuevo objetivo.

OBJETIVO : $y_1 = \xi_1$

HECHOS :

...

(10) $y_1 \in [0,1]$

(11) $f_1(y_1) = y_1$

Este problema se resuelve utilizando el teorema del valor medio, que se incluye en el sistema en forma de la siguiente regla:

$$f \textit{ derivable en } [a,b] \ \wedge x, y \in [a,b]$$
$$\Longrightarrow \exists z \in [a,b] \textit{ tal que } f(x) - f(y) = f'(z)\,(x - y)$$

Las condiciones de la regla se cumplen en el intervalo $[0,1]$ con los hechos (2), (8) y (10), dando lugar a la igualdad:

(12) $f_1(y_1) - f_1(\xi_1) = f_1'(z_1)\,(y_1 - \xi_1)$

Ahora la metaregla de resolución de ecuaciones para demostrar igualdades extrae el sistema formado por (9), (11) y (12) y los términos ξ_1, y_1, $f_1(\xi_1)$, $f_1(y_1)$ y $f_1'(z_1)$. Las rutinas de resolución de ecuaciones proporcionan la solución del sistema. La instanciación del hecho universal (3) con la constante atómica z_1 descarta la solución en que $f'(z_1) = 1$, de forma que se obtiene $\xi_1 = y_1$, que se convierte en regla de reescritura que hace que el objetivo se reescriba como True.

Queda el tercer subobjetivo, que consiste en comprobar que el valor obtenido para x pertenece a $[0,1]$, es decir, $\xi_1 \in [0,1]$, que se satisface inmediatamente por (8). Un Matemático probablemente dedicaría poca atención a este aspecto, pero PROGENES debe comprobar que todo es correcto. Habiendo sido resueltos los tres subproblemas, el problema queda terminado.

□

4.5.5 Problema 9

Problema 9. *Demostrar que toda función lipschitziana de* $\mathbb{R}$ *en* $\mathbb{R}$ *es continua en* $\mathbb{R}$.

Este problema se puede resolver sin utilizar la definición de límite en términos de ε y δ. Sabiendo que $|f(x)-f(y)| \leq M|x-y|$, eliminando el valor absoluto en el primer miembro y despejando $f(x)$, tenemos $-M|x-y|+f(y) \leq f(x) \leq M|x-y|+f(y)$. Tomando límite en $x \to y$, $f(y) \leq \lim_{x\to y} f(x) \leq f(y)$, luego $\lim_{x\to y} f(x) = f(y)$, es decir, f es continua en y para todo $y \in \mathbb{R}$.

A primera vista esta demostración parece asequible para PROGENES. Sin embargo, el sistema no se da cuenta de la conveniencia de despejar $f(x)$ en la desigualdad, y por lo tanto no llega a aplicar la metaregla de introducción de operadores en desigualdades, así que no es capaz de obtener la demostración anterior. Como consecuencia, PROGENES resuelve el problema utilizando la definición de límite en términos de ε y δ. Cabe preguntarse cuál de las dos demostraciones es más natural. Lo cierto es que ambas son perfectamente válidas en este sentido. Es muy posible que un matemático o un estudiante utilizasen la definición, ya que la relación $|f(x)-f(y)| \leq M|x-y|$ puede sugerir inmediatamente una forma de acotar la diferencia de imágenes. Sin embargo, si uno prefiere evitar la definición de límite, probablemente realizaría la otra demostración.

Veamos la demostración de PROGENES.

OBJETIVO : f_1 *continua en* $\mathbb{R}$

HECHOS : (1) f_1 *lipschitziana*

Tras fracasar los intentos de resolver el problema utilizando teoremas, se aplica la definición de los metapredicados `continua` y `lipschitziana`, de manera que después de eliminar la cuantificación existencial de la hipótesis ($\exists M > 0 \ldots$) y la universal del objetivo ($\forall a \in \mathbb{R} \ldots$), obtenemos:

OBJETIVO : $\lim_{t \to a_1} f_1(t) = f_1(a_1)$

HECHOS :

(1') $\forall x, y \in \mathbb{R},\ |f_1(x) - f_1(y)| \leq M_1 |x - y|$
(2) $M_1 > 0$

En este problema omitiremos la lista de constantes, que no influye en la demostración.

El sistema intenta nuevamente sin éxito aplicar diversas heurísticas, como la de introducción de operadores en desigualdades. Finalmente, agotadas las posibilidades de aplicar reglas, se activa la definición procedural de `limite` en términos de ε y δ:

OBJETIVO : $\forall \varepsilon > 0, \exists \delta > 0$ *tal que* $\forall t,\ |t - a_1| < \delta \Rightarrow |f(t) - f(a_1)| < \varepsilon$

Eliminando cuantificadores,

OBJETIVO : $(\forall t,\ |t - a_1| < \delta \Rightarrow |f(t) - f(a_1)| < \varepsilon)\ \wedge\ \delta > 0$

INCOGNITAS : $\delta \in \mathbb{R}$

HECHOS :

...
(3) $\varepsilon_1 > 0$

El sistema empieza por el primer subobjetivo, de forma que, skolemizando y eliminando la implicación, tenemos:

OBJETIVO : $|f(t_1) - f(a_1)| < \varepsilon_1$

HECHOS :

...
(4) $|t_1 - a_1| < \delta$

La regla de introducción de operadores se aplica al hecho (1') respecto del objetivo, con el operador identidad y $\sigma \rightarrow [t_1/x, a_1/y]$. Se obtiene:

(5) $|f(t_1) - f(a_1)| \leq M_1|t_1 - a_1|$

después de lo cual se puede aplicar la metaregla basada en la transitividad de la desigualdad, descrita en el apartado 4.4.3, creando un subproblema en el que se plantea el subobjetivo:

OBJETIVO : $M_1|t_1 - a_1| < \varepsilon_1$

Entonces, utilizando la regla general de introducción de operadores (teniendo en cuenta que (2) establece que $M_1 > 0$), el sistema multiplica por M_1 los dos miembros del hecho (4), cuyo primer miembro unifica con una subexpresión del objetivo, obteniendo:

(6) $M_1|t_1 - a_1| < M_1\delta$

Aplicando nuevamente la transitividad de las desigualdades, se plantea el subproblema:

OBJETIVO : $M_1\delta \leq \epsilon_1$

Teniendo en cuenta que δ es una variable cuantificada existencialmente y el objetivo es una desigualdad, el sistema llama a un módulo de solubilidad que da la solución $\delta \rightarrow \frac{\varepsilon_1}{M_1}$. El funcionamiento de este módulo se explica detalladamente en [SAI94].

El subobjetivo pendiente $\delta > 0$ sería como mucho una mera comprobación rutinaria para un matemático. PROGENES necesita utilizar algunas reglas. Una vez substituido en el objetivo el valor obtenido para δ, el problema consiste en demostrar que $\frac{\varepsilon_1}{M_1} > 0$. Aplicando la regla general de introducción de operadores, y teniendo en cuenta que $M_1 > 0$, se dividen por M_1 los dos miembros de la desigualdad $\varepsilon > 0$ (hecho (3)), obteniendo lo que buscábamos.

□

Capítulo 5

Perspectivas en el campo de la enseñanza

Como ya se ha apuntado en otros capítulos, dadas sus características, PROGENES es un sistema apropiado para su utilización en el contexto educativo, al estar basado en un modelo cognitivo de un experto en el dominio [BAZ93, CAS93, CAS93b]. En este capítulo se discutirá la utilidad del sistema en este sentido, y se planteará su posible integración en un sistema tutor inteligente. No se trata de presentar una aplicación plenamente desarrollada, sino de dar una idea lo que se puede hacer con nuestro sistema en el campo de la enseñanza, presentando el modelo PROGENES bajo una perspectiva didáctica.

El desarrollo de un tutor inteligente para un dominio específico requiere un programa experto capaz de resolver problemas del dominio, un modelo del estudiante, y un modelo pedagógico [SLE82]. Los campos científicos como las Matemáticas o la Física requieren una aproximación basada en la Deducción Automática. Los sistemas de producción han demostrado ser una herramienta útil, aunque costosa, para la modelización de este tipo de conocimiento [AND90].

Por otra parte, el conocimiento relacionado con estos campos científicos tiene una componente procedural que juega un papel fundamental en la aplicación del mismo. Una tarea esencial en este sentido es la descripción del conocimiento procedural de tal forma que acorte y direccione efectivamente las demostraciones y refleje etapas típicas en la resolución de problemas. Los métodos que se han propuesto para ello están basados en reglas de producción que representan metaconocimiento que aplica

el conocimiento deductivo [AND85]; las reglas de reescritura se han utilizado de forma limitada [AND90], pero no como parte del modelo. Sin embargo, parte del conocimiento procedural relacionado con una disciplina científica simplemente nos dice que algunos objetos o hechos que son iguales a otros deben ser siempre representados por estos últimos. Y parte de este conocimiento se adquiere mientras se resuelve el problema. La modelización de este conocimiento sobre la igualdad puede simplificar el desarrollo de un tutor, y permite una mejor interacción con el estudiante.

Como ya se indicó en el capítulo 3, la terminología relativa a lo procedural y lo declarativo no se ajusta a un criterio fijo universalmente aceptado. En general, en los ámbitos próximos a las ciencias cognitivas, y en particular, en el campo de la enseñanza asistida por ordenador, se suele otorgar la condición de conocimiento procedural a las reglas de producción. Para evitar confusiones, utilizaremos aquí la denominación de *conocimiento deductivo* para referirnos a este tipo de conocimiento, y *conocimiento procedural de aplicación directa* cuando nos refiramos al conocimiento programado que se puede ejecutar de forma inmediata con una simple llamada.

En este capítulo se describirá un modelo del conocimiento relacionado con el campo de la Mecánica elemental, a un nivel de primer curso de enseñanza media. Las principales características de este modelo son su simplicidad conceptual (todos los conceptos y descripciones del conocimiento se clasifican de una manera simple y limpia), y el hecho de que el conocimiento procedural de aplicación directa se incorpora mediante el sistema de reescritura de PROGENES, que cambia dinámicamente y se integra con el sistema de producción subyacente.

Nuestro modelo está basado en la representación del conocimiento descrita en el capítulo 3, e incluye por tanto una descripción de los objetos relacionados con el tema que se estudia (en nuestro caso, instantes, partículas, etc.), formas de referirse a estos objetos (posición de una partícula, etc.), afirmaciones acerca de los mismos, y reglas que representan distintos tipos de conocimiento. Las reglas de producción, como ya se dijo en el capítulo 3, representan conocimiento de tipo declarativo o deductivo, mientras que las metareglas tienen un carácter más procedural. El conocimiento procedural propiamente dicho se representa por medio de reglas de reescritura. El tipo de problemas para los que está diseñado el modelo, así como la modelización del dominio, son similares a los del sistema MECHO [BUN79] (ver capítulo 2).

Nuestro modelo permite la interacción entre la máquina y el estudiante en relación con las razones de la incapacidad de éste para resolver un problema, de manera más rica de lo que permiten otros modelos previos. La riqueza de la interacción que permite un

modelo es una medida parcial de la calidad del mismo. Más concretamente, si nuestro modelo se utiliza en una implementación de tal manera que las reglas de producción y las de reescritura son objetos per se sobre los que el sistema puede razonar y que puede comparar con el *input* del estudiante, se puede lograr una interacción altamente satisfactoria. Finalmente, es esencial utilizar una interfaz flexible y potente que pueda ser adaptada a diferentes circunstancias y tipos de interacción. El desarrollo de estas interfaces es más factible hoy en día gracias al desarrollo de herramientas basadas en modelos [SZE92], que permiten el diseño interactivo de una gran parte de la interfaz, así como un prototipado rápido, con un refinamiento sucesivo de diseños parciales.

En la primera sección de este capítulo se describe con detalle el modelo propuesto, utilizando un problema como ejemplo. En la sección siguiente se comentarán algunos aspectos más técnicos relacionados con las características del sistema PROGENES, que sirve de soporte para la implementación del modelo. En la sección 5.3 se ilustrarán brevemente las posibilidades de interacción del modelo.

5.1 El modelo

En esta sección describiremos un modelo de los conocimientos de Mecánica elemental que un estudiante debe adquirir para resolver problemas en dicho dominio. Para ello estudiaremos con detalle un ejemplo, lo cual nos permitirá ofrecer una visión del conocimiento que está ligado dinámicamente al proceso de resolución de un problema. Los objetos y relaciones que describimos aquí, y que son relevantes para el ejemplo, aparecen en problemas generales de Mecánica a un nivel elemental. La descripción del modelo viene a ser una paráfrasis de los aspectos descritos en capítulos anteriores, orientada a un tratamiento didáctico de la materia considerada.

El problema que vamos a estudiar es el siguiente:

> **Problema 15.** *Un vehículo parte del punto A con velocidad constante de 60 km/h. Después de 180 minutos se detiene durante una hora, tras lo cual, vuelve hacia el punto de salida con aceleración uniforme, llegando a A cuatro horas después. ¿Cuál es la aceleración del vehículo en el camino de vuelta?*

Este problema es sólo un ejemplo sencillo pero apropiado para la descripción de

los conceptos que queremos introducir. Nuestro modelo está pensado para problemas generales sobre movimiento de cuerpos, en los que pueden aparecer varias partículas con distintos tipos de movimiento, fuerzas, etc. Se trata de un modelo de la evolución del conocimiento en distintas etapas de este proceso. Empezaremos por describir los aspectos estáticos del proceso. Esta parte del modelo incluye las siguientes componentes:

- Objetos. En el ejemplo aparecen un vehículo, cuatro instantes y dos puntos. Los objetos constituyen una jerarquía. Por ejemplo, los vehículos se clasifican de acuerdo con el tipo de movimiento que tienen y la dimensión del espacio en que se mueven, y las magnitudes se clasifican según su tipo en relación con las magnitudes básicas de espacio y tiempo.

- Denotaciones de objetos: metafunciones. En el ejemplo, las distintas posiciones del vehículo en los cuatro instantes denotan puntos; las velocidades del vehículo durante los dos primeros intervalos de tiempo denotan magnitudes de tipo *espacio/tiempo*, igual que la velocidad del vehículo en cualquier instante del tercer intervalo; la aceleración del vehículo durante el tercer intervalo denota una magnitud de tipo $espacio/tiempo^2$. Durante la resolución del problema, uno debe utilizar o bien distancias entre pares de puntos, o bien las coordenadas de los puntos para denotar números; otros problemas pueden incluir otras denotaciones como velocidad media, vector velocidad, o vector aceleración.

- Relaciones entre los objetos: metapredicados. En el ejemplo, la velocidad del vehículo durante el primer intervalo de tiempo es de 60 km/h, y las posiciones inicial y final del vehículo son iguales al punto A; además de la igualdad, pueden aparecer otras relaciones, como la de aproximación de una partícula a un punto durante un intervalo de tiempo. También tenemos leyes genéricas, principios y teoremas de Mecánica y Geometría, que se pueden representar en forma de reglas, y definiciones de metafunciones. Por ejemplo, el espacio recorrido por un vehículo durante un intervalo de tiempo se puede definir como la distancia entre las posiciones inicial y final del vehículo si éste sigue un movimiento uniforme, mientras que en otros casos la definición es más compleja.

Nuestro modelo incluye la descripción de conocimiento procedural de aplicación inmediata. Un ejemplo de este tipo de conocimiento es la conversión de magnitudes a un sistema de unidades fijo, habitualmente el Sistema Internacional. Esto es una de las primeras cosas que un estudiante debería hacer en un problema como el que

estamos analizando, y es un proceso en el que típicamente se suele aplicar conocimiento procedural, que consiste en un procedimiento fijo que el estudiante ha aprendido y que le permite obtener la magnitud requerida normalizada. Por ejemplo, el estudiante puede substituir cualquier unidad que aparezca en cualquier magnitud por su equivalente en el Sistema Internacional, y realizar entonces manipulaciones simbólicas formales. De esta manera, suponiendo que tiene el conocimiento adecuado sobre cómputos simbólicos, puede convertir unidades de la misma manera que puede sumar números o vectores, o realizar cómputos matemáticos más complejos.

En nuestro modelo podemos describir mecanismos procedurales que pueden justificar que dos cosas son lo mismo simplemente porque sabemos lo que una representa, y la substituimos siempre por la otra. Esto nos permite tener un mayor control de lo que el estudiante sabe o se supone que sabe en un momento dado. De esta forma, la parte del modelo que dirige el proceso de resolución se descompone en dos dimensiones independientes, deducción y substitución, lo que hace que el control sea mucho más simple.

Un estudiante típicamente incorpora una gran cantidad de conocimiento en esta forma procedural. De hecho, una componente esencial del proceso de aprendizaje consiste en la asimilación de este conocimiento: el conocimiento procedural directo evoluciona a lo largo del tiempo. Habitualmente lo que en un determinado estadio del aprendizaje es conocimiento declarativo se convierte en conocimiento procedural de aplicación inmediata en un estadio posterior. Esto se puede observar incluso en libros de Mecánica de diferentes niveles. Por ejemplo, un libro de mayor nivel utilizará el conocimiento procedural de que siempre que la posición de un vehículo está determinada en todo instante, su aceleración viene dada por la segunda derivada de la posición respecto del tiempo. En cambio los libros introductorios no pueden utilizar este conocimiento, puesto que el estudiante no conoce aún los conceptos matemáticos correspondientes.

Por otra parte, el conocimiento procedural puede tener distintos niveles de calidad. Por ejemplo, muchos estudiantes aprenden de memoria las fórmulas de conversión a unidades internacionales para un determinado tipo de magnitudes, como velocidades (utilizando por ejemplo, que $1m/s = 3.6km/h$). Este conocimiento procedural es de una calidad mucho más baja que el descrito anteriormente, ya que no puede ser utilizado para la conversión de otras magnitudes, como por ejemplo aceleraciones. La calidad del conocimiento procedural de una base de conocimiento se puede medir en términos de dos dimensiones fundamentales: su nivel de abstracción y el ahorro de tiempo y esfuerzo que permite en el proceso del razonamiento.

Nuestro modelo incluye definiciones de objetos y denotaciones. Por ejemplo, el espacio recorrido por un vehículo durante un intervalo de tiempo es igual a la distancia entre su posición inicial y final, si tiene movimiento uniforme. Este conocimiento se introdujo más arriba como si fuese declarativo, pero en la práctica se trata de conocimiento procedural de aplicación directa, ya que en cualquier problema donde un vehículo tiene movimiento uniforme y aparece el espacio que recorre en un intervalo de tiempo, éste se substituye por la definición correspondiente. Este conocimiento procedural puede ser mejorado definiéndolo para cualquier tipo de movimiento, no necesariamente uniforme, siempre que el vehículo no de la vuelta hacia la dirección opuesta durante el intervalo de tiempo considerado. Esta mejora requiere la definición de la relación de *'dar la vuelta'*, que se cumple cuando la velocidad se anula en un determinado instante sin que se anule la aceleración.

Veamos la resolución del problema. En primer lugar, cuando leemos el enunciado adquirimos inmediatamente un conocimiento procedural. En este caso, sabemos que la velocidad del vehículo durante el primer intervalo de tiempo es de $50/3\ m/s$. Puesto que se trata de un valor totalmente determinado, se crea el conocimiento procedural que establece que siempre que nos refiramos a esta velocidad, la substituimos por $50/3\ m/s$. Lo mismo ocurre con la posición de la partícula en el primer y último instante, que es igual a A.

La regla general que está detrás de todo esto es la heurística de proceduralización de igualdades descrita en la sección 4.3.1 del capítulo anterior. Recordemos que según esta heurística, cuando se sabe (en forma declarativa) que dos objetos son iguales y que uno de ellos es canónico, entonces creamos el conocimiento procedural de aplicación directa (regla de reescritura) según el cual el otro objeto debe ser substituido en todas partes por el objeto canónico.

La regla para derivar conocimiento procedural a partir de igualdades se utiliza tan pronto como se satisfacen sus condiciones en cualquier momento durante la resolución del problema.

Volvamos a nuestro ejemplo. Por simplificar la explicación, utilizaremos los símbolos t_0, t_1, t_2 y t_3 para referirnos a los instantes relevantes consecutivos del problema, y x para representar el vehículo. El primer paso de la resolución está basado en la ley fundamental del movimiento uniforme, que dice que el espacio recorrido durante un intervalo de tiempo es igual al producto de la velocidad por el tiempo transcurrido. En el ejemplo la ley se puede aplicar en los dos primeros intervalos, dando lugar a las dos igualdades correspondientes:

$$\text{espacio\_recorrido}\,(x, t_0, t_1) = (t_1 - t_0)\ \text{velocidad}\,(x, t_0, t_1)$$
$$\text{espacio\_recorrido}\,(x, t_1, t_2) = (t_2 - t_1)\ \text{velocidad}\,(x, t_1, t_2)$$

En este caso, el conocimiento procedural asociado al espacio recorrido por un vehículo con movimiento uniforme hace que el primer miembro de las igualdades de arriba se substituya por la distancia entre las posiciones inicial y final:

$$\text{distancia}\,(\text{posicion}\,(x, t_0), \text{posicion}\,(x, t_1)) = (t_1 - t_0)\ \text{velocidad}\,(x, t_0, t_1)$$
$$\text{distancia}\,(\text{posicion}\,(x, t_1), \text{posicion}\,(x, t_2)) = (t_2 - t_1)\ \text{velocidad}\,(x, t_1, t_2)$$

Por otra parte, el problema nos da valores determinados para varias de las expresiones que aparecen en las igualdades anteriores. Estos datos consisten en igualdades que la regla de derivación de conocimiento procedural convierte automáticamente en reglas de reescritura. Aplicando estas reglas, las igualdades de arriba se convierten en:

$$\text{distancia}\,(A, \text{posicion}\,(x, t_1)) = 180000\ m$$
$$\text{distancia}\,(\text{posicion}\,(x, t_1), \text{posicion}\,(x, t_2)) = 0\ m$$

que se convierten a su vez en reglas de reescritura. Por otra parte, una ley de geometría establece (de forma procedural) que el hecho de que la distancia entre dos puntos sea 0 se substituye automáticamente por la igualdad entre los dos puntos, de forma que la última de las igualdades anteriores da lugar a la igualdad:

$$\text{posicion}\,(x, t_1) = \text{posicion}\,(x, t_2) \tag{1}$$

que representa conocimiento puramente declarativo.

La ley del movimiento uniformemente acelerado se puede aplicar en el tercer intervalo, una vez que se demuestra que el vehículo no da la vuelta durante dicho intervalo. En consecuencia, utilizando el conocimiento procedural derivado anteriormente, se obtiene:

$$\text{distancia}\,(\text{posicion}\,(x, t_2), A) = 103680000\ \text{aceleracion}\,(x, t_2, t_3)\ s^2 \tag{2}$$

Ahora se aplica una regla para la derivación de conocimiento declarativo a partir de igualdades, que es la regla de introducción de operadores descrita en el capítulo anterior, que al aparecer el segundo miembro de (1) en el primer miembro de (2), establece una nueva igualdad:

$$\text{distancia}\,(\text{posicion}\,(x, t_2), A) = \text{distancia}\,(\text{posicion}\,(x, t_1), A)$$

Utilizando la regla de reescritura asociada al segundo miembro (junto con el conocimiento procedural que nos dice que la distancia y la igualdad son conmutativos), se obtiene:

$$\text{distancia}\,(\text{posicion}\,(x, t_2), A) := 180000\ m$$

donde una vez más hemos utilizado la regla de proceduralización de la igualdad. Aplicando esta regla de reescritura en la igualdad (2), tenemos:

$$180000\ m = 103680000\ \text{aceleracion}\,(x, t_2, t_3)\ s^2$$

Finalmente, nuestro modelo incluye metarazonamiento por medio de metareglas que involucran el plan de demostración (como utilizar o no coordenadas), o el tipo de objetivo y objetos que aparecen en el problema. En nuestro caso, se aplica la regla de resolución de ecuaciones descrita en el capítulo anterior (sección 4.4.2), obteniendo:

$$\text{aceleracion}\,(m, t_2, t_3) = \frac{1}{576}\ m/s^2$$

Este tipo de problemas se pueden resolver también utilizando coordenadas, para lo cual es necesario definir un sistema de coordenadas, fijando un origen y un vector que defina la dirección positiva (en el caso de un espacio de dimensión 1). Esto se puede hacer proceduralmente utilizando como origen la posición inicial de una de las partículas del problema y tomando el sentido de su vector velocidad como orientación positiva. Se utilizarían otras denotaciones para las coordenadas, y se aplicarían otras reglas, pero no diferirían esencialmente en cuanto a los aspectos del modelo que estarían involucrados.

5.2 Aspectos técnicos

El modelo descrito encaja muy bien en las características del sistema PROGENES. El sistema proporciona un lenguaje de representación de objetos que incluye mecanismos de diseño orientado a objetos, y permite manejar el tipo de entidades que aparecen en problemas como el anterior. Asimismo, PROGENES tiene la capacidad deductiva necesaria para la aplicación de reglas y metareglas. Por último, el lenguaje PROGENES proporciona el soporte simbólico necesario para la representación del conocimiento procedural de aplicación directa, incluyendo la creación dinámica de reglas de reescritura.

Asimismo el lenguaje PROGENES permite manejar expresiones con niveles arbitrario de anidamiento, lo cual se ajusta a la necesidad de manejar expresiones complejas del tipo de las que pueden aparecer en los problemas.

Finalmente, la posibilidad de hacer referencia en las metareglas a los elementos del problema por su aspecto o contenido permite la implementación de mecanismos como la derivación de conocimiento procedural a partir de igualdades, o la obtención de nuevas igualdades que involucran objetos que aparecen en otras igualdades (introducción de operadores).

Como resolvedor automático de problemas, PROGENES controla automáticamente todos estos mecanismos. El sistema es quien decide el momento adecuado para poner en marcha uno u otro. En un entorno interactivo, por el contrario, se debe permitir que el usuario tome el control de la resolución de los problemas, y por tanto se deben desactivar los mecanismos que actualmente rigen el comportamiento de PROGENES. Por ejemplo, el usuario puede decidir aplicar un teorema en dirección backward, a pesar de ser posible hacerlo en forward. Aunque PROGENES no hace backward hasta que no sea posible utilizar razonamiento forward, el usuario debe tener la libertad de hacerlo, mientras sea correcto.

El módulo experto debe contener en su estructura interna los mecanismos y el conocimiento necesarios para realizar las mismas deducciones que puede hacer el estudiante, de forma que cuando éste da un paso en la resolución de un problema, el sistema pueda comprobar a qué operación corresponde en su representación interna, y qué implicaciones tiene. Para ello los procedimientos que permiten a PROGENES controlar su propia actividad deban ser substituidos por la utilización de descripciones declarativas del tipo de mecanismos que el modelo permite aplicar. Se trata de que el sistema pueda seguir el razonamiento del estudiante, interpretando correctamente sus

intenciones, y corrigiendo en su caso los errores cometidos. Los mecanismos de control de PROGENES tienen una utilidad a pesar de todo, a la hora de prestar prestar ayuda o consejo al usuario.

5.3 Interacción

En esta sección presentaremos brevemente las posibilidades de interacción que permite nuestro modelo. La descripción no pretende ser un estudio profundo de estas posibilidades, sino simplemente poner en evidencia el hecho de que una aplicación eficiente e interesante es posible a partir del modelo de PROGENES.

El primer aspecto fundamental de la interacción es el *input* y *output* de objetos y denotaciones. Nuestra representación del conocimiento procedural de aplicación directa permite un tratamiento de este aspecto más apropiado que en otros sistemas utilizados previamente, si se utiliza en conjunción con interfaces sensibles al contexto [SZE92], con características como la adaptación dinámica al contexto y la generación automática de ayuda [MOR93]. Estas interfaces permiten la representación de objetos y denotaciones en la forma prescrita por el conocimiento procedural que ha sido derivado en cada instante, así como su actualización automática.

La existencia de una clasificación de objetos relativamente compacta facilita tanto su representación gráfica como la referencia a objetos específicos. Por ejemplo, el usuario puede determinar un punto indicando mediante el ratón en pantalla un punto de la recta que representa el espacio donde el movimiento tiene lugar, y respondiendo determinadas preguntas en un menú, o bien a través de la barra de menú principal, seleccionando *'punto'*, y determinando el resto de la información a través de otro menú.

Un segundo aspecto fundamental de la interacción es permitir que el estudiante pregunte por información de tipo más general. Aparte del tipo de información que el usuario puede requerir en otros tutores, nuestro modelo permite preguntar por los hechos relevantes que debe tener en mente de forma directamente procedural, pero que puede haber olvidado. Entre otras, el usuario puede formular preguntas del tipo siguiente:

- *"¿Cómo puedo resolver este problema?"*
  (La respuesta puede ser *"utilizando coordenadas"*, *"sin coordenadas"*, o ambas)

- *"¿Qué conozco sobre los puntos que aparecen en el problema? ¿Qué sé de este punto concreto?"*

  En general, el estudiante puede preguntar lo que sabe (o debe saber) sobre los objetos de un tipo determinado, o sobre objetos que satisfacen determinada relación, como por ejemplo, las partículas que se mueven en la misma dirección que una partícula dada. La respuesta se presenta en un formato distinto para los diferentes tipos de conocimiento, incluyendo el procedural directo.

- *"¿Cuáles son las velocidades relevantes en este problema? ¿Cuáles son conocidas?"*

  En general, se pueden hacer estas preguntas sobre cualquier denotación. La segunda de ellas pregunta por el conocimiento procedural generado durante la resolución del problema.

- *"Ayuda (¿qué puedo hacer ahora?)"*

  La respuesta a esta pregunta será cada vez más detallada si se repite varias veces consecutivas. En un principio el sistema puede intentar inducir al estudiante a utilizar determinado conocimiento procedural que debería ya conocer, o en un segundo estadio, otros tipos de conocimiento. Posteriormente, el sistema puede dar ciertas indicaciones sobre el siguiente paso a tomar, o incluso instrucciones concretas sobre qué hacer.

- *"Dame una representación gráfica del problema. Incluye en el gráfico el conocimiento procedural derivado hasta el momento."*

  Estos requerimientos pueden tener también diferentes niveles de respuesta.

- *"¿Qué tipo de movimiento tiene esta partícula?"*

  Esto puede ser útil en problemas donde las partículas están sujetas a fuerzas, como cuando una partícula se mueve por un plano inclinado.

Por otra parte, el programa puede también hacer preguntas al estudiante, de acuerdo con distintas motivaciones: a) a petición de éste, para hacerle practicar; b) para pedirle información como consecuencia de un primer requerimiento de ayuda; c) para pedirle detalles sobre un determinado paso que ha dado en la resolución del problema, aunque sea correcto (por ejemplo, la primera vez que el estudiante utiliza coordenadas, el sistema tratará de asegurarse de que sabe bien cuál es el sistema de coordenadas que ha tomado y sus consecuencias); d) para tratar de corregirle en caso de que haya cometido un error.

La máquina puede formular ciertas preguntas al estudiante una vez que el problema ha sido resuelto de forma interactiva, como por ejemplo, preguntarle sobre el significado de las velocidades o aceleraciones negativas en caso de que aparezcan en la solución, o requerir que resuelva el problema utilizando un sistema de coordenadas distinto o aplicando un método diferente.

Un aspecto importante en la interacción es la decisión sobre los planes para la resolución del problema, como la utilización de coordenadas. Esto involucra dos aspectos: el estudiante debe poder preguntar al ordenador sobre posibles planes, y éste tiene que ser capaz de deducir a partir de los pasos que da el usuario cuál es el plan que está siguiendo. El primer aspecto es mucho más sencillo que el segundo. Una buena aproximación para el segundo puede ser la utilización de metaconocimiento ad hoc, en forma de reglas sencillas pero útiles, como por ejemplo:

Si P es un plan
y R es una regla
y el hecho de que P este activo es una de las condiciones de R
y P no esta activado
y todas las demas condiciones de R son conocidas del estudiante
y el estudiante afirma la conclusion de la regla
Entonces el estudiante esta activando el plan P.

Por ejemplo, las reglas que introducen las coordenadas de los vectores incluyen entre sus condiciones que el plan del problema debe ser el de utilizar coordenadas; por otra parte, las reglas que involucran distancias entre puntos incluyen entre sus condiciones que el plan del problema debe ser utilizar cómputos escalares. De modo que cuando el estudiante pretende realizar una acción que aparece en la conclusión de estas reglas, y todas las condiciones de las mismas se cumplen excepto la relativa al plan, el sistema deduce que el estudiante está siguiendo dicho plan.

Conociendo los planes del problema, el ordenador puede o bien ayudar al estudiante a tomar decisiones sobre ellos, o aconsejarle cuando está dando pasos que corresponden a planes incompatibles, lo cual se considera como un comportamiento errático.

Quizás el aspecto más importante de la interacción del usuario con la máquina en este contexto es que hace que el estudiante resuelva los problemas de forma coherente, siguiendo un camino consistente. En esta dirección, el sistema filtra la información que le da el estudiante sobre los hechos que demuestra, y le hace retroceder al paso anterior

si está haciendo algo en el momento equivocado. Por ejemplo, se requiere al estudiante que escriba todas las magnitudes que aparecen en el problema utilizando unidades del Sistema Internacional antes de hacer cualquier otra cosa. Después de esto, el usuario deberá determinar el tipo de movimiento de cada partícula, y entonces deberá dibujar un esquema gráfico de los movimientos de las partículas.

Los párrafos anteriores no son de ninguna manera una descripción de las capacidades interactivas de nuestro modelo. Un estudio completo requeriría el desarrollo de un modelo del conocimiento del estudiante, incluyendo inferencias sobre errores típicos, y por tanto sale del ámbito del presente trabajo. El desarrollo de un entorno interactivo está en este momento en fase de diseño, y los aspectos discutidos en este capítulo pretenden dar una idea de la riqueza de interacción que permite el modelo del dominio contenido en PROGENES.

Capítulo 6

Conclusiones

Es difícil concebir la resolución de problemas de un cierto grado de complejidad sin la capacidad de razonar sobre el aspecto del problema, sobre lo que sabemos y lo que buscamos, o sobre la posibilidad de considerar distintas estrategias [PIT90]. Para lograr un avance significativo en el campo de la resolución automática de problemas es necesario que el sistema sea capaz de controlar y dirigir las deducciones que hace, y en este sentido creemos que es fundamental la utilización de técnicas basadas en metaconocimiento, de uso extendido hoy día en el campo de los sistemas expertos [DAV82], y llevadas también con éxito a la demostración de teoremas en sistemas como MUSCADET [PAS89]. PROGENES pretende contribuir al avance en esta dirección, mediante el desarrollo de técnicas suficientemente generales como para que su utilidad no se limite exclusivamente al tratamiento de un dominio muy restringido.

Por otra parte, en la distancia que separa las posibilidades de los sistemas desarrollados de la capacidad humana, un aspecto esencial reside en la gran flexibilidad del ser humano para combinar cálculos y deducciones, conocimiento sobre el *qué* y conocimiento sobre el *cómo*, es decir, conocimiento declarativo y conocimiento procedural. A la vez que la posibilidad de emplear métodos procedurales predefinidos permite acelerar la obtención de resultados, proporciona también mayores y más potentes medios para utilizar metaconocimiento.

Una gran parte del metaconocimiento utilizado en PROGENES tiene que ver con la aplicación de los axiomas de la igualdad, lo cual está en relación directa con los paradigmas de *evaluación* procedural [BOY88]. Este metaconocimiento responde al propósito de realizar una aplicación rápida y efectiva, al tiempo que bien encaminada,

de estos axiomas. Así por ejemplo, las propiedades simétrica y transitiva de la igualdad son aplicadas automáticamente por el propio motor de inferencia, mediante un algoritmo de unificación extendido. La propiedad reflexiva está implementada proceduralmente en el código de la metafunción =. Los axiomas de substitución se aplican por medio de la definición de metafunciones evaluables, la creación de reglas de reescritura a partir de igualdades, y la introducción heurística de operadores en igualdades. Asímismo, la resolución de ecuaciones aplica implícitamente todas estas propiedades de la igualdad, tanto las que le corresponden como relación de equivalencia, como las de substitutividad (al substituir las soluciones en el problema). Si medimos la calidad del metaconocimiento por la generalidad y el rango de utilidad del mismo, junto con la proximidad a la fundamentación axiomática subyacente, podemos decir que la calidad del metaconocimiento que PROGENES contiene es buena en este sentido.

Además del metaconocimiento de utilidad general, también es importante definir heurísticas más particulares para el manejo de conceptos específicos como funciones, números reales, etc., dado que el conocimiento matemático no es homogéneo, en el sentido de que en la práctica habitual de las Matemáticas, no todos los conceptos se tratan de la misma manera.

La representación del metaconocimiento en PROGENES no es tan uniforme como en MUSCADET, donde todo el metaconocimiento se codifica en forma de metareglas. En nuestro sistema, una parte fundamental del metaconocimiento está representado en forma de metareglas. Sin embargo, el motor de inferencia contiene también metaconocimiento muy general, como son las heurísticas de descomposición de problemas, la instanciación de hechos universales por objetos, el tratamiento de la igualdad, o la activación de metafunciones. Asímismo, el usuario puede plasmar metaconocimiento mediante la representación direccionada de definiciones y propiedades de conceptos en forma de metafunciones. Todas estas heurísticas se podrían haber implementado por medio de metareglas. Sin embargo, de esta manera se consigue una mayor riqueza expresiva y una utilización eficiente del conocimiento, aunque en contrapartida la complejidad de los mecanismos es mayor, y se pierde en modularidad. Probablemente, esta representación menos homogénea es más parecida a la que utiliza una persona.

Las técnicas presentadas en este trabajo permiten un tratamiento eficiente de una clase de problemas que es interesante de por sí. La utilidad de las técnicas desarrolladas va más allá del dominio que hemos considerado inicialmente, y en particular han mostrado un alto grado de adecuación en áreas como la Mecánica elemental (ver [CAS92, CAS94, SAI94]). El sistema proporciona soluciones naturales, y por lo tanto fáciles de entender para una persona. Esto facilita la comunicación entre el sistema y

un potencial usuario, lo cual hace que PROGENES tenga también interés de cara a su adaptación en aplicaciones relacionadas con la enseñanza [BAZ93, CAS93, CAS93b].

Perspectivas

Para terminar, indicaremos en esta sección algunas posibles direcciones de continuación de nuestro trabajo. Por una parte, el trabajo realizado se puede completar con el desarrollo de nuevas bases de conocimiento para el tratamiento de otros dominios, el diseño de nuevas heurísticas a partir del estudio de los métodos y estrategias habituales en el trabajo de los matemáticos, o el desarrollo de mecanismos (más específicos quizás) que permitan un control más preciso de la activación de las metafunciones. Por otra, existen posibilidades de ampliación o mejora del sistema, que se apuntarán a continuación, en aspectos como la flexibilidad de la representación del conocimiento, la compilación del conocimiento, o la formalización axiomática profunda de los mecanismos desarrollados. Por último, las perspectivas de adaptación de nuestro trabajo para la implementación de un tutor inteligente sido ya presentada con detalle en el capítulo 5.

La versatilidad de la representación del conocimiento es una característica fundamental en la la actividad del experto, y es un aspecto importante también de cara a conseguir que el sistema tenga una cierta generalidad. Como hemos visto, PROGENES incluye mecanismos que permiten obtener en parte esta flexibilidad, integrados en un paradigma orientado a objetos, de forma que la representación de los objetos se puede adecuar a los requerimientos del problema a resolver.

Además del tipo de casos que hemos descrito, existen otros que PROGENES no contempla. Por ejemplo, la continuidad de una función en espacios métricos tiene varias definiciones equivalentes. Los matemáticos utilizan una u otra según el caso, y de esta elección puede depender en gran medida el grado de complicación y dificultad de la demostración. No sería difícil permitir que el sistema utilizase todas las definiciones a la vez, pero sería más interesante desarrollar heurísticas que permitan a PROGENES decidir cuál de ellas es más conveniente.

Por otra parte, no hay una sola representación posible mediante objetos y metafunciones para un dominio dado. Por ejemplo, una recta se puede representar por distintos tipos de ecuaciones. Un muelle se puede describir por su constante de elasticidad, la posición de un extremo, y su elongación, de forma que la tensión que soporta se calcularía a partir de la constante y la elongación. Pero también se podría incluir la tensión entre las componentes del muelle, definiendo la elongación como función de

la constante y la tensión. Asímismo, en los libros de Matemáticas de enseñanza primaria, se dice que dos rectas en el plano son paralelas si no se cortan en ningún punto. Aunque esta definición puede ser apropiada para una comprensión intuitiva del concepto, sobre todo si no se dispone aún de las nociones elementales de Algebra Lineal, resulta poco manejable a la hora de resolver problemas de Geometría Analítica. Es por ello que en PROGENES la definición procedural del metapredicado `paralelo` consiste en comprobar si los vectores normales a sendas rectas son proporcionales, cuando se dispone de las ecuaciones de ambas. Por último, cuando se define un predicado, cabe también la posibilidad de representarlo mediante un tipo, es decir, en lugar de definir un metapredicado como `continua`, se podría construir el tipo de las funciones continuas, imponiendo en la definición del mismo la condición de que el límite de la función en todo punto sea igual a su imagen en dicho punto.

La persona que desarrolla las bases de conocimiento es quien decide cuál es la forma más apropiada de representar los conceptos del dominio, salvo en algunos casos, como la proceduralización automática de igualdades. Sería interesante seguir trabajando en la dirección del desarrollo de mecanismos que permitan al sistema realizar la compilación de determinados tipos de conocimiento.

Por otra parte, el trabajo en PROGENES se ha centrado en el desarrollo del resolvedor en sí, y no se han abordado aún cuestiones como la presentación de los resultados al usuario, de forma que la utilización del sistema exige un conocimiento previo del funcionamiento y los principios del mismo. En este sentido está previsto el desarrollo de una interfaz que facilite la comunicación con el usuario y ofrezca una traza más amigable de la resolución de los problemas, incorporando explicaciones, mostrando únicamente la información relevante, y filtrando operaciones de rutina, como operaciones aritméticas o simplficaciones algebraicas elementales.

Finalmente, el desarrollo de las técnicas descritas en este trabajo se ha llevado a cabo partiendo del estudio de la resolución de problemas propios de libros de texto de Cálculo y Geometría de primeros cursos de universidad [SPI81, SAL86, DEM78, HER87, PAI67, POL65], que presentan el dominio bajo un enfoque eminentemente práctico. Esto nos ha permitido orientar nuestro trabajo de forma muy directa hacia los objetivos que perseguimos alcanzar, subordinando el desarrollo de modelos teóricos a los resultados prácticos deseados. Una vez establecidos los mecanismos y fundamentos principales del sistema, un estudio en profundidad de las materias tratadas, a través de otro tipo de textos que realicen un tratamiento más exhaustivo de las mismas, partiendo de dominios básicos como la Teoría de Conjuntos [GOD67], permitirá reforzar la fundamentación teórica de las técnicas desarrolladas.

Bibliografía

[AND83] Anderson J. R., *The Architecture of Cognition*, Harvard University Press, Cambridge, Massachusets, 1983.

[AND85] Anderson J. R., *Cognitive Psychology and its Implications*, 2nd edition, W. H. Freeman and Company, New York, 1985.

[AND90] Anderson J. R., Boyle C. F., Corbett A. T. y Lewis M. W., *Cognitive Modeling and Intelligent Tutoring*, Artificial Intelligence 42, 1990, pp 7-49.

[BAZ93] Bazin J. M., Castells P., Moriyón R., Saiz F., *A Knowledge Based Problem Solver Conceived for Intelligent Tutoring Applications*, Proceedings ICCTE 93, Kiev, September 1993.

[BAZ93b] Bazin J. M., *GEOMUS: un résolveur de problèmes de géométrie qui mobilise ses connaissances en fonction du problème posé*, Thèse de doctorat de l'Université Paris VI, 1993.

[BLE71] Bledsoe W. W., *Splitting and reduction heuristics in automatic theorem proving*, Artificial Inelligence 2, 1971, pp. 55-77.

[BLE72] Bledsoe W. W., Boyer R. S., Henneman W. H., *Computer Proofs of Limit Theorems*, Artificial Intelligence, volume 3, 1972, pp. 27-60.

[BLE77] Bledsoe W. W., *Non-resolution Theorem Proving*, Artificial Intelligence 9, 1977, pp. 1-35.

[BOY79] Boyer R. S., Moore J S., *A Computational Logic*, Academic Press, Inc., 1979.

[BOY88] Boyer R. S., Moore J S., *A Computational Logic Handbook*, Academic Press, Inc., 1988.

[BUC84] Buchanan G., Shortlife E., *Rule-based Expert Sistems: The MYCIN Experiments of the Standford Heuristic Program Project*, Addison-Wesley, 1984.

[BUN79] Bundy A., Byrd L., Luger G., Mellish C., Palmer M., *Solving mechanics problems using meta-level inference*, IJCAI, 1979, pp. 1017-1027.

[CAS91] Castells P., Díaz J., Moriyón R., Rodriguez-Marín P. and Saiz F., *PROGENES: Una Base de Conocimiento para la resolucion de problemas de Calculo Infinitesimal*, Proceedings AEPIA 91, Madrid, 1991, pp. 17-30.

[CAS92] Castells P., Díaz J., Gonzalo J.,Moriyón R., Rodriguez-Marín P., Saiz F., Tobar M.J., *A Scientific Problem Solver with a Natural Language Interface*, Proceedings ISCIS 92, Presses EHEI, Paris, 1992, pp.237-243.

[CAS92b] Castells P., Díaz J., Gonzalo J.,Moriyón R., Rodriguez-Marín P., Saiz F., Tobar M.J., *Un sistema para la resolucion de problemas enunciados en Lenguaje Natural*, Actas PRODE 92, Universidad Politécnica de Madros, 1992, pp. 232-245.

[CAS93] Castells P., Moriyón R., Saiz F., Villa E. , *Application of Techiques of Automatic Problem Solving to Computer Intelligent Tutoring*, Proceedings ICCTE 93, Kiev, September 1993.

[CAS93b] Castells P., Moriyón R., Saiz F. Villa E., *A Model of Knowledge in Elementary Mechanics*, Proceedings ICCE 93, Taiwan, December 1993.

[CAS93c] Castells P., Moriyón R., Saiz F., *PROGENES: using metaknowledge to solve scientific problems*, Informe técnico IIC, 1993.

[CAS93d] Castells P., Moriyón R., Saiz F., *A goal-based reasoning heuristic in automatic problem solving*, Informe técnico IIC, 1993.

[CAS93e] Castells P., Moriyón R., Saiz F., *Jerarquías de objetos en la resolución de problemas científicos*, Actas CAEPIA 93, Madrid, 1993, pp. 89-98.

[CAS94] Castells P., Moriyón R., Saiz F., *PROGENES: an automated problema solver based on Mathematica*, Informe técnico IIC, 1994.

[CHA70] Chang C. L., *The unit proof and the input proof in theorem proving*, J. ACM 17, 1970, pp. 698-707.

[CLA93] Clarke E., Zhao X., *ANALYTICA: A Theorem Prover for Mathematica*, The Mathematica Journal, Vol. 3, Issue 1, Winter 1993, pp. 56-71.

[DAV80] Davis R., *Meta-rules: Reasoning about control*, Artificial Intelligence, volume 15, 1980, pp. 179-222.

[DAV80b] Davis R., *Content reference: Reasoning about rules*, Artificial Intelligence, volume 15, 1980, pp. 223-239.

[DAV82] Davis R., Lenat D. B., *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.

[DEM78] Demidovitch B., *Problemas y ejercicios de Análisis Matemático*, Ed. Paraninfo, Madrid, 1978.

[FIN91] Fink P., *The Role of Domain Knowledge in the Design of an Intelligent Tutoring System*, in Burns H., Parlett J. W., Luckhardt C., *Intelligent Tutoring Systems, Evolutions in Design*, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1991.

[GOD67] Godement R., *Algebra*, Editorial Tecnos S.A., Madrid, 1967.

[HER87] Hernández E., *Algebra y Geometría*, Ediciones de la Universidad Autónoma de Madrid, 1987.

[LEN83] Lenat D. B., *EURISKO: A program that learns new heuristics and domain concepts. The nature of heuristics III: Program design and results*, Artificial Intelligence, volume 21, 1983, pp. 61-98.

[MCA89] Mcallester, D. A., *Ontic: a knowledge representation system for Mathematics*, Massachussets Institute of Technology, 1989.

[MOR94] Moriyón, R., *Automatic Generation of Semantic Help for User Interfaces*, Proceedings CHI 94, 1994.

[PAI67] Paige L. J., Swift S. D., *Elementos de Algebra*, Ed. Reverté S.A., Barcelona, 1967.

[PAS84] Pastre D., *MUSCADET: Un système de démonstration automatique de théorèmes utilisant connaissances et métaconnaissances en mathématiques*, Thèse d'état de l'Université Paris VI, 1984.

[PAS89] Pastre D., *An Automatic Theorem Proving System Using Knowledge and Metaknowledge*, Artificial Intelligence, volume 38, 1989, pp. 257-318.

[PAU87] Paulson L. C., *Logic and computation*, Cambridge University Press, Cambridge, 1987.

[PIT90] Pitrat J., *Métaconnaissance, futur de l'intelligence artificielle*, Hermes, Paris, 1990.

[POL88] Polson & Richardson (eds.), *Foundations of Intelligent Turoring Systems*, Lawrence Erlbaum, Hillsdale, New Jersey, 1988.

[POL65] Polya G., *Mathematical Discovery (or understanding, learning, and teaching problem solving)*, vols. I and II., John Wiley & Sons, Inc., New York, 1965.

[ROB65] Robinson J. A., *A machine-oriented logic based on the resolution principle*, J. ACM 12, 1965, pp.23-41.

[ROU75] Roussel Ph., *PROLOG: Manuel de référence et d'utilisation*, Groupe d'Intelligence Artificielle, Marseille-Lumigny, 1975.

[SAI94] Saiz F., *Fundamentación de un sistema en resolución automática de problemas*, Tesis de doctorado de la Universidad Autónoma de Madrid Departamento de Ingeniería Informática, Marzo 1994.

[SAL86] Salas S. L., Hille H., *Calculus de una y varias variables con Geometría Analítica*, Editorial Reverté, S.A., Barcelona, 1986.

[SZE92] Szekely P., Luo P., Neches R., *Facilitating the exploration of interface design alternatives: the Humanoid model of interface design*, Proceedings SIGCHI 92, 1992, pp. 507-515.

[SHO76] Shortlife E., *Computer-based Medical Consultations: MYCIN*, American Elsevier, New York, 1976.

[SLE82] Sleeman D., Brown J. S. (eds.), *Intelligent Tutoring Systems*, Academic Press, New York, 1982.

[SPI81] Spivak M., *Calculus*, W. A. Benjamin Inc., New York, 1981.

[WOL91] Wolfram S., *Mathematica, A System for Doing Mathematics by Computer*, Wolfram Research, Inc., 1991.

[WOS65] Wos L., Robinson G., Carson D., *Efficiency and completeness of the set of support strategy in theorem proving*, J. ACM 12, 1965, pp. 536-541.